Durben · Löffelmann
Plenge · Vüllers

COMMODORE



Das große GRAFIK-Buch

EIN DATA BECKER BUCH

DAS STEHT DRIN:

Der Commodore 128 steht seinem kleinen Bruder, dem C-64, in bezug auf Grafik in nichts nach – das beweist dieses Buch. Ein Team von Grafik-Spezialisten deckt wirklich alle Geheimnisse des C-128 auf. Mit diesem Buch können Sie sofort alle Möglichkeiten Ihres Commodore-Rechners für eigene Programme nutzen!

Aus dem Inhalt:

- Die 3 Modi des C-128
- Grafikbefehle im 128er-Modus
- Betriebsarten des VIC-Chips
- Verwaltung der HI-RES
- Lage des Grafikspeichers
- Farbgebung in der HRG/MC-Grafik
- Programmierung von Sprites
- IMR/IRQ-Interruptprogrammierung
- Der Lightpen
- Register des VDC-Chips
- Aufbau des Video-RAM
- Der Charactergenerator
- HI-RES-Grafik mit dem VDC
- Animationsgrafiken
- Statistische Auswertungen
- Funktionsplotter
- Einführung in CAD
- Ein/Ausgabe von Grafiken
- Grafikprogrammierung in 6502-Assembler

UND GESCHRIEBEN HABEN DIESES BUCH:

Axel Plenge, bekannter DATA BECKER-Autor des Grafikbuches zum C-64, ist es gelungen, mit Klaus Löffelmann und Dieter Vüllers, die Abiturienten am Schloß-Gymnasium-Overhagen in Lippstadt sind, und Ralf Durben, Informatikstudent, ein Team von erfahrenen Grafikspezialisten auf die Beine zu stellen.

ISBN 3-89011-154-8



Durben · Löffelmann
Plenge · Vüllers

COMMODORE



Das große GRAFIK-Buch

EIN DATA BECKER BUCH

ISBN 3-89011-154-8

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.



Inhaltsverzeichnis

I. Einführung

1.1 Einleitung 11
1.2 Die drei Modi des Commodore 128 14
1.3 Die Grafikbefehle im 128er Modus 16
1.3.1 Hochauflösende Multicolor-Grafik 17
1.3.2 Die Textgrafikbefehle 38
1.3.3 Sprite/Shape-Grafik 47

II. Der VIC II (8566 II)

2.1 Einführung zum VIC II 63
2.2 Grundsätzliches zum VIC II 64
2.3 Registerbeschreibung des 8566 69
2.4 Die Betriebsarten 74
2.4.1.1 Die Verwaltung der HI-RES 75
2.4.1.2 Die Lage des Grafikspeichers 81
2.4.1.3 Die Farbgebung in der HRG 83
2.4.1.4 Die Farbgebung und Punktsetzung in der MC 88
2.4.2 Die Sprites 93
2.4.2.1 Form, Lage und Verwaltung der Sprites 94
2.4.2.2 Positionierung der Sprites 97
2.4.2.3 Aktivieren der Sprites 99
2.4.2.4 Farbgebung bei Sprites 101
2.4.2.5 Sprite-Prioritäten 102
2.4.2.6 Vergrößerung der Sprites 105
2.4.2.7 Kollisionen 106
2.4.3 Der Textmodus 108
2.4.3.1 Die Zeichensätze 109
2.4.3.2 Verschiebung des Zeichensatzes 111
2.4.3.3 Modifikation des Zeichensatzes 116
2.4.3.4 Der Extended-Color-Modus 120

2.4.3.5 Der Multi-Color-Modus 123
2.4.3.6 CHAR-Designer (40 Zeichenmodus) 124
2.4.3.7 Verschieben des Textschirms 139
2.5 IMR/IRR-Interruptprogrammierung 144
2.5.1 5 * VIC II 148
2.5.2 Speed up mit dem VIC II 160
2.6 Scrolling 164
2.7 Der Lightpen 166
2.8 Das Farb-RAM 170

III. Der VDC (8563)

3.1 Der VDC 173
3.2 Registerbeschreibung des 8546 175
3.3 Grundsätzliches zum VDC 180
3.4 VDC-Utilities 182
3.4.1 Das Assembler-Listing 183
3.4.2 Der BASIC-Lader 184
3.4.3 Utilities-Erklärung 186
3.5 Die Arbeitsregister des VDC 188
3.6.1 Der Aufbau des Video-RAM 192
3.6.2 Das Attribute-RAM 194
3.6.3 Der Charactergenerator 196
3.6.4 Der Grafikspeicher 197
3.7 Adreßänderungen 198
3.8 VDC CHAR-Designer 200
3.9.1 VDC HI-RES-Grafik 213
3.9.2 VDC HI-RES-Multi-Color-Grafik 219
3.10.1 Bildschirmformat Änderung 222
3.10.2 Alles über den Cursor 224
3.10.3 Das Blinken und Unterstreichen von Zeichen 226

IV. Grundlagen der Grafikprogrammierung

4.1 Linie 227
4.2 Das Polarkoordinatensystem 231
4.3.1 Parallelprojektion 235
4.3.2 Zentralprojektion 240

V. Anwendungen der Grafikprogrammierung

5.1 Unterhaltungsgrafik 245
5.1.1 Laufschriften 245
5.1.2 Sprite-Animation 247
5.1.3 Etwas über Spiele 249

5.2 Statistik 251
5.2.1 Die 2-D-Balkengrafik 256
5.2.2 Die 3-D-Balkengrafik 262
5.2.3 Die Kuchengrafik 267
5.2.4 Die Explosions-Kuchengrafik 276
5.2.5 Gewinn-Verlust-Grafiken (Vergleichsgrafiken) 284
5.2.6 Kurvengrafik 290

5.3 Funktionsplotter 294
5.3.1 2-D-Funktionsplotter 294
5.3.2 3-D-Funktionsplotter 299

2.4.3.5 Der Multi-Color-Modus 123
2.4.3.6 CHAR-Designer (40 Zeichenmodus) 124
2.4.3.7 Verschieben des Textschirms 139
2.5 IMR/IRR-Interruptprogrammierung 144
2.5.1 5 * VIC II 148
2.5.2 Speed up mit dem VIC II 160
2.6 Scrolling 164
2.7 Der Lightpen 166
2.8 Das Farb-RAM 170

III. Der VDC (8563)

3.1 Der VDC 173
3.2 Registerbeschreibung des 8546 175
3.3 Grundsätzliches zum VDC 180
3.4 VDC-Utilities 182
3.4.1 Das Assembler-Listing 183
3.4.2 Der BASIC-Lader 184
3.4.3 Utilities-Erklärung 186
3.5 Die Arbeitsregister des VDC 188
3.6.1 Der Aufbau des Video-RAM 192
3.6.2 Das Attribute-RAM 194
3.6.3 Der Charactergenerator 196
3.6.4 Der Grafikspeicher 197
3.7 Adreßänderungen 198
3.8 VDC CHAR-Designer 200
3.9.1 VDC HI-RES-Grafik 213
3.9.2 VDC HI-RES-Multi-Color-Grafik 219
3.10.1 Bildschirmformat Änderung 222
3.10.2 Alles über den Cursor 224
3.10.3 Das Blinken und Unterstreichen von Zeichen 226

IV. Grundlagen der Grafikprogrammierung

4.1 Linie 227
4.2 Das Polarkoordinatensystem 231
4.3.1 Parallelprojektion 235
4.3.2 Zentralprojektion 240

V. Anwendungen der Grafikprogrammierung

5.1 Unterhaltungsgrafik 245
5.1.1 Laufschriften 245
5.1.2 Sprite-Animation 247
5.1.3 Etwas über Spiele 249

5.2 Statistik 251
5.2.1 Die 2-D-Balkengrafik 256
5.2.2 Die 3-D-Balkengrafik 262
5.2.3 Die Kuchengrafik 267
5.2.4 Die Explosions-Kuchengrafik 276
5.2.5 Gewinn-Verlust-Grafiken (Vergleichsgrafiken) 284
5.2.6 Kurvengrafik 290

5.3 Funktionsplotter 294
5.3.1 2-D-Funktionsplotter 294
5.3.2 3-D-Funktionsplotter 299

5.4 Computer Aided Design (CAD) 309
5.4.1 Einführung in CAD 309
5.4.2 Einteilung in verschiedene Ebenen 310
5.4.3 Das Menue 311
5.4.4 Das Dialogprogramm 314
5.4.5 Die I/O-Operationen 317
5.4.6 Die Grafikroutinen 319
5.4.7 Das gesamte CAD-System 323

5.5 Ein-/Ausgabe Grafiken 331
5.5.1 Hardcopy RX80/FX80 40 Zeichen 331
5.5.2 Hardcopy RX80/FX80 80 Zeichen 336

5.6 Grafikprogrammierung in 6502-Assembler 341
5.6.1 Supergrafik 128 341
5.6.2 Assemblerlisting 342
5.6.3 BASIC-Lader 356

ANHANG

-A- Tabellierung aller Escape-Funktionen 361
-B- Tabellierung aller Control-Funktionen 363
-C- ASCII- und Bildschirmcodetabelle 364
-D- Quellenverweise 368

I. Einführung

1.1 Einleitung

"Dieser Computer muß ja eine super Maschine sein".

So oder ähnlich reagierten wohl auch Sie, als die ersten Testberichte oder Werbeanzeigen, wie:

"Bad news for IBM and Apple"

erschienen.
Zugegeben, ein Computer, der eigentlich aus dreien besteht, ist auch etwas Außergewöhnliches. Dazu kam weiterhin die Tatsache, daß diesem Gerät hervorragende Grafikeigenschaften zugesprochen wurden. - Grund genug, uns näher mit dem "Neuen" von Commodore auseinanderzusetzen.
Das Ergebnis unserer Arbeit liegt vor Ihnen: Das Grafikbuch zum Commodore 128.

Bevor wir nun tiefer "in medias res" dringen wollen, lassen Sie uns zunächst etwas Grundsätzliches über den Commodore 128 sagen. Er war, so glauben wir, eines der seltenen Geräte, die schon vor ihrer Veröffentlichung einen hohen Bekanntheitsgrad hatten. Dies mag bestimmt mehreren Fakten zugeschrieben werden, jedoch waren nach unserer Meinung der Hauptgrund für diese schnelle Berühmtheit die grafischen Fähigkeiten (gutes BASIC, hohe Auflösung, 2 Videocontroller, 80 und 40 Zeichen usw.).

Aus all diesen Gründen reizte uns dieses Gerät eben so sehr, wie manch anderen unter Ihnen auch, und es entstand das Grafikbuch zum Commdore 128. Da der Rechner dem Commodore 64, seinem erfolgreichen Vorgänger, in einigen Punkten ähnlich blieb, sind wir auf die Features, die schon das 64er Grafikbuch auf ausführlichste Weise beschreibt, nicht erschöpfend

eingegangen. Natürlich werden Ihnen neben den vom 64er bekannten, die besonderen Features des Commodore 128 ausführlich erläutert, und die neu hinzugekommenen oder im 64er Grafikbuch vernachlässigten Themen größtenteils noch mit gut dokumentierten BASIC-Beispielprogrammen versehen.

Das Buch haben wir in vier große Abschnitte unterteilt:
Der erste Abschnitt liefert eine Einführung, die das Arbeiten mit Ihrem neuen Gerät stark vereinfacht. Sie können dort alle Grafikbefehle des Commodore 128 nachlesen. Wir haben diese Befehle bewußt nochmals aufgeführt, weil das Handbuch, soviel besser es auch gegenüber dem des C-64 auch sein mag, einige Mängel in den Erklärungen so mancher Befehle aufweist.

Der zweite Abschnitt befaßt sich dann mit dem 8566-(VIC II) Grafikprozessor. Hier können Sie wirklich alles über diesen IC und seine Handhabung herausfinden. Der VIC 8566, ein Nachfolger des 6569, der ja schon im C 64 seine Anwendung fand, ist um zwei Register erweitert und auch in seinen restlichen Funktionen total überarbeitet worden. Deshalb werden wir nochmals dort auf alle Einzelheiten des VIC II eingehen. Auch wird dort eine Routine zu finden sein, die den 128er im 64er Modus um 25% schneller macht - und das ohne Einschränkung.

Ein weiterer Grafikprozessor, der jedoch ganz neu konstruiert wurde, wird in Kapitel 3 genauestens beschrieben. Sie werden an diesem Kapitel bestimmt Freude haben, denn wer programmiert nicht gerne mit einer Auflösung von 640 * 200 Punkten, und das in 16 Farben.

Für all diejenigen unter Ihnen, denen es nicht reicht, eine Linie mit DRAW auf den Bildschirm zu zeichnen, oder für die, die wissen wollen, wie man einen Kreis "per pedes" berechnet, haben wir das 4. Kapitel geschrieben. Dort werden alle mathematischen Grundlagen zur Berechnung verschiedener geometrischer Gebilde Schritt für Schritt erarbeitet, so daß auch alle "Nicht-Einsteiner" auf diesem Gebiet der Informatik Fuß fassen können.

Im fünften Kapitel kommen dann die Anwendergenies unter Ihnen voll auf ihre Kosten. Dort lernen Sie, wie Sie den Computer im Bereich der Grafik auch für gewerbliche oder schulische Sachen nutzen können. Dies ist ein Gebiet der Grafikprogrammierung, das immer populärer wird. Es lohnt sich also. Themen, die dort behandelt werden, sind: Laufschriften, Statistik, CAD, Druckerroutinen und vieles mehr.

Soweit nun unsere kurze Einführung. Doch nun lassen Sie uns mit dem ersten Abschnitt beginnen, der das Mysterium 'Drei Computer in einem' behandelt und Ihnen einen grundsätzlichen Überblick über Ihren Computer geben soll.

Achtung: **In den Listings kann aus drucktechnischen Gründen der ↑ als ^ erscheinen!**

1.2 Die drei Modi des Commodore 128

Wie oben schon gesagt, besteht der Commodore 128 ja eigentlich aus drei Computern, die in einem Gehäuse zusammengefaßt sind. Der erste Computer repräsentiert den Commodore 64, der zweite den eigentlichen Commodore 128 und der dritte einen vollwertigen CP/M-Computer. Wie ist das nun zu verstehen? - Nun, im Grunde genommen ist das ganz einfach. Ihr Computer besitzt natürlich nur eine Tastatur, und Sie brauchen für ihn selbstverständlich keine drei Diskettenlaufwerke. Der einzige Unterschied zu 'Single-Computern' ist die Tatsache, daß im Commodore 128 mehrere Betriebssysteme integriert wurden. Das eine Betriebssystem simuliert einen 64er in allen Funktionen, das zweite enthält alle Befehle und Funktionen des BASICs 7.0 (s.u.) und das dritte, das von der Diskette geladen werden muß, umfaßt alle CP/M-Systemoperationen.

Wir wollen uns hier ausschließlich mit dem zweiten Betriebssystem, also mit dem Commodore 7.0 BASIC, beschäftigen, da dieses sehr leistungsfähige Befehle zur Grafik- und Spriteprogrammierung bietet. Wer dennoch, aus welchen Gründen auch immer, lieber die Grafik im 64er-Modus ansehen möchte, dem sei an dieser Stelle das Grafikbuch zum Commodore 64, das ja auch auf speziellere Hardwareeigenschaften des Gerätes eingeht, wärmstens empfohlen.

Jetzt bliebe nur noch zu klären, wieso es möglich ist, einem normalen Computer drei Betriebssysteme 'einzupflanzen'. Das Problem lösten die Commodore-Entwickler, indem sie ihrer neuen Schöpfung ganz einfach zwei Prozessoren mitgaben. Der eine Prozessor, ein 8502, der voll kompatibel zum 6510 ist, kann wahlweise mit einem oder zwei Megaherz getaktet werden, und der andere Prozessor, ein Z80, wird, man lese und staune, mit 4 MHz getaktet. Soweit nun die groben technischen Details für alle Hardware-Freunde. Anfänger sollten sich nicht durch die vielen Fachbegriffe wild machen lassen, sie dienen nur den interessierten 'Hardwarefreaks'. Ein Problem, dessen Lösung auch für alle Nicht-Hardwarespezialisten interessant sein dürften, ist die Aktivierung des Z80-Prozessors. Dieses Problem

ist ebenfalls sehr einfach zu lösen, denn Sie brauchen (fast) gar nichts zu machen - nur den Computer einzuschalten. Dann nämlich macht sich der Z80-Prozessor das erste Mal bemerkbar und versucht, das CP/M-Betriebssystem von der Diskette in den Computer zu laden. Findet er das Betriebssystem nicht, so aktiviert der Z80-Prozessor den 8502, der dann wiederum die Interpretation des BASIC 7.0 übernimmt. Hierzu noch ein Tip am Rande: Legen Sie, wenn Sie den Commodore 128 und ein 1541-Laufwerk besitzen, immer eine Diskette in das Laufwerk. Unterließen Sie dieses, so hätte das denselben Effekt wie das Entfernen einer Diskette während des Ladens. Mit anderen Worten, das Floppylaufwerk führt einen Bump aus (nicht zu überhören) und versucht erneut, das CP/M-Programm zu booten. Da dieser Bump, nicht wie beim 1570/1571 Laufwerk, durch eine Lichtschranke ersetzt wird, sondern rein mechanischer Natur ist, kann es vorkommen (und nach einiger Zeit wird es auch vorkommen), daß sich der Tonkopf Ihres Laufwerkes durch das ständige Anschlagen an die hintere Schienenbegrenzung verstellt. Ersparen Sie sich also unnötige Kosten.

Wir wollen uns hier nun, wie oben schon gesagt, mit dem zweiten Modus befassen, welcher selbst wiederum in zwei verschiedene Modi unterteilt ist.
Der Commodore 128 bietet Ihnen nämlich die Besonderheit, über eine 80- und eine 40-Zeichen-Darstellung zu verfügen. Es ist klar, daß hierzu zwei verschiedene Videocontroller (elektronische Bauteile, die für den Bildschirmaufbau zuständig sind) vorhanden sein müssen. Allerdings ergeben sich hierbei leider auch einige Nachteile: Der Videocontroller, der für die 40 Zeichendarstellung verantwortlich ist, hat nur einen Composite Ausgang. Sie können an diesem nur einen handelsüblichen Fernseher (weil nur dieser Ausgang HF-moduliert wird) oder einen Monitor, wie den Commodore 1701 anschließen. 80 Zeichen können Sie jedoch nur auf einem RGB-Monitor betrachten. Lösung wäre hier der neue Commodore 1902-Monitor, der beide Eingänge besitzt. Diesen können Sie dann mit einem kleinen Umschalter am Rücken des Gerätes jeweils auf die gewünschte Betriebsart einstellen.

Soweit nun die grundsätzlichen Eigenschaften im 128er-Modus. Wenn es Ihnen bis hierher etwas verwirrend vorkam, so darf ich Sie beruhigen. In unseren Hardwarekapiteln, die sich mit beiden Videocontrollern befassen, werden diese Sachen in weitaus ausführlicherem Maße dargestellt, jedoch müssen bei diesem Computer seine "Außergewöhnlichkeiten" sofort geklärt werden, um so mit kleinstem Aufwand, die bestmögliche Effektivität zu erreichen.

* Unter Taktfrequenz versteht man - grob ausgedrückt - die Anzahl der Grundfunktionen, die der Prozessor in einer Sekunde ausführen kann. Beim 6502 z.B. wären das rund eine Million Operationen in einer Sekunde (!!!).

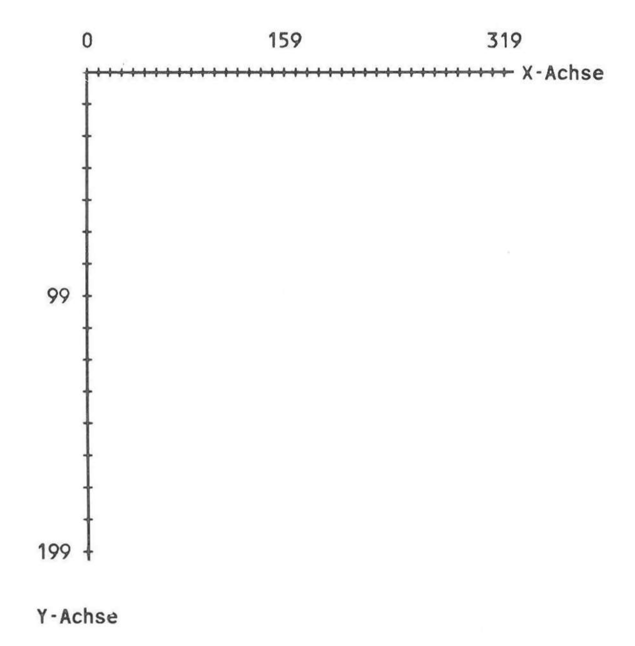
1.3 Die Grafikbefehle im 128er-Modus

Bestimmt werden Sie sich jetzt wundern, wieso an dieser Stelle die Grafikbefehle des Commodore 128 aufgezählt werden, die ja ausnahmsweise alle im Handbuch zum C-128 so schön erklärt wurden. Lassen wir dazu Radio Eriwan antworten: Im Prinzip haben Sie recht, jedoch sind in diesem Handbuch, so gut es gegenüber dem 64er Handbuch auch sein mag, die Befehlsbeschreibungen nicht immer richtig, so daß zum vernünftigen Arbeiten, keine gute Grundlage vorhanden gewesen wäre. Weiterhin meinen wir, daß es nicht sehr sinnvoll ist, wenn Sie immer zwischen Commodore Handbuch und Grafikbuch hin und herblättern müssen, wenn Sie selber mal irgendein Grafikprogramm schreiben möchten, und Sie sich noch nicht so gut mit den Befehle auskennen. In der Aufmachung haben wir uns stark an das Commodore-Handbuch gehalten um Sie nicht all zu sehr zu verwirren.

Wir hoffen jetzt, daß unser Grafikbuch teilweise auch an die Stelle des Commodore-Handbuches treten wird, und wünschen Ihnen viel Freude bei der Erarbeitung der Grafikbefehle.

1.3.1 Hochauflösende/Multicolor-Grafik

Das Ziel eines interessierten Grafikprogrammierers ist es wohl immer, schöne Bilder zu erzeugen. Diese Bilder sollten möglichst exakt kleinere Einzelheiten wiedergeben. Das ist aber mit der normalen Textgrafik nicht möglich, da die Bildschirmpunkte entweder zu groß sind oder die Anzahl der verschiedenen Zeichen begrenzt ist. Um diese Mängel zu beseitigen, gibt es die hochauflösende Grafik. Der C-128 ist in der Lage, den Bildschirm in ein Raster von 200 Zeilen und 320 Spalten zu unterteilen. Zur Verdeutlichung dient dieses Schaubild:



In der Multicolor-Grafik halbiert sich die Anzahl der horizontalen Punkte, also der Punkte, die auf der X-Achse liegen. Dafür stehen dann insgesamt vier Farben zur Verfügung.

Die nun folgenden Befehle befassen sich mit diesem Koordinatensystem. Sie sollten, bevor Sie nun mit Eifer diese Befehle ausprobieren, zunächst alles über den Befehl "GRAPHIC" nachlesen, und die folgenden Dinge beherzigen:

Sie können grundsätzlich die Parameter eines Grafikbefehls, die Sie nicht benötigen, weglassen. In der Regel wird für diesen Parameter dann der Wert "0" eingesetzt. Anstelle dieses Parameters muß dann ein Komma gesetzt werden.

Die BOX-Anweisung:

```
Format     : BOX fq,x1,y1,x2,y2,wn,au

Parameter : - fq    : Farbquelle.
                     0=Hintergrund
                     1=Vordergrund
                     2=Zusatzfarbe 1
                     3=Zusatzfarbe 2
                     Der Wert definiert die Quelle der
                     Zeichenfarbe

            - x1,y1 : Koordinaten des linken, oberen
                      Eckpunktes
                      x1: HGR 0-319    MC 0-159
                      y1: HGR 0-199    MC 0-199

            - x2,y2 : Koordinaten des rechten, unteren
                      Eckpunktes. Wertebereiche wie x1
                      bzw. y1

            - wn    : Winkel, um den das Rechteck
                      gedreht werden soll (in Grad).
                      Voreingestellt ist der Wert 0.

            - au    : Legt fest, ob das Reckteck ausge-
                      malt werden soll (1), oder nicht
                      (0).

Beispiel    : BOX 1,10,10,100,100,45,1

Funktion    : Zeichnen eines Rechtecks
```

Erläuterungen:

Dieser Befehl schließt gleich mehrere Optionen ein: Das Rechteck ist ausfüllbar und läßt sich drehen. Die in den Parametern angegebene Farbquelle gibt praktisch an, welche Farbe benutzt werden soll. Die Definition der einzelnen Farbquellen bewirkt der COLOR-Befehl. Dort werden die Quellen auch erklärt.

Beispielsprogramm:

```
 10 rem *****************
 20 rem **programm box**
 30 rem *****************
 40 color 0,1              :rem hintergrundfarbe
 50 color 4,1              :rem rahmenfarbe
 60 color 1,3              :rem punktfarbe
 70 graphic 1,1            :rem grafikseite einschalten
 80 w=0                    :rem drehwinkel
 90 x1=10 : y1=10          :rem linke, obere eckkoordinaten
100 x2=190 : y2=190        :rem rechte, untere eckkoordinaten
110 w=w+2                  :rem aenderung des drehwinkels
120 box 1,x1,y1,x2,y2,w,0  :rem rechteck zeichnen
130 x1=x1+5 : y1=y1+5      :rem aenderung der eckpunkte
140 x2=x2-3 : y2=y2-3      :rem aenderung der eckpunkte
150 if x1=120 then 130     :rem ende der grafik
160 goto 110
170 goto 170
```

Der CIRCLE-Befehl

Format : CIRCLE fq,x,y,xr,yr,st,en,wn,sw

Parameter : - fq : Farbquelle (siehe COLOR)

- x,y : Koordinaten des Mittelpunktes (scaliert)

- xr : Radius der Hauptachse (scaliert)

- yr : Radius der Nebenachse (scaliert) Voreingestellt ist der Wert xr

- st : Startwinkel (in Grad). Voreingestellt ist der Wert 0

- en : Endwinkel (in Grad). Voreingestellt ist der Wert 360

- wn : Winkel, um den die Figur gedreht werden soll (in Grad) Voreingestellt ist der Wert 0

- sw : Gibt den Winkelabstand zwischen zwei Berechnungspunkten an. Wertebereich: 1-255 -je kleiner der Wert, desto runder die Figur. Voreingestellt ist der Wert 2

Beispiel : CIRCLE 1,100,100,60,30,,,40,3

Funktion : Zeichnen einer Ellipse, eines Kreises oder eines bestimmten Polygons.

Erläuterungen:

Der CIRCLE-Befehl ist sehr vielfältig. So ist es nicht nur möglich, Kreise oder Ellipsen zu zeichnen. Sogar Polygone sind mit diesem leistungsfähigen Befehl kein Problem. Hierzu ist der Parameter sw sehr wichtig. Je kleiner dieser Wert, desto kreisähnlicher das Gebilde. Die untenstehende Tabelle gibt über die konkreten Werte für diesen Parameter zur Polygonzeichnung mehr Aufschluß. Falls Sie noch nichts über scalierte Eingaben wissen, so lesen Sie bitte unter dem SCALE-Befehl nach.

Beispielprogramm:

```
100 rem ************************************
110 rem *** circle-demo ******************
120 rem ************************************
130 :
140 graphic1,1
150 for x=1 to 120 step 3
160 :  circle 1,160,100,100,100,,, x,120
165 :  rem  mod, xm, ym, xr, yr,,,dw, sw
170 next x
180 :
190 graphic 1,1
200 for x=1 to 180 step 1.5
210 :  circle 1,160,100,100-(x/2),100-(x/2),,, x,x+30
215 :  rem  mod, xm, ym, x-radius, y-radius,,,dw, sw
220 next
```

Polygon	Seitenwert
Dreieck	120
Viereck	90
Fünfeck	72
Sechseck	60
Zehneck	36
Zwölfeck	30

Allgemein können Sie die Anzahl der Ecken eines Polygons durch folgende Formel errechnen:

Anzahl der Ecken = 360 / sw

Der CHAR-Befehl

```
Format       : CHAR fq,x,y,z$,iv

Parameter    : - fq   : Farbquelle
                        0=Hintergrundfarbe
                        1=Vordergrundfarbe
                        2=Zusatzfarbe 1
                        3=Zusatzfarbe 2

               - x,y  : Koordinaten des Textanfangs.
                        x: 0-39 bzw. 79
                        y: 0-24

               - z$   : Text, der ausgegeben werden soll

               - iv   : 0=Normale Darstellung
                        1=Inverse Darstellung

Beispiel      : CHAR 1,3,20,"Der CHAR-Befehl"

Funktion      : Plaziert einen Text auf dem Grafik-
                Bildschirm.
```

Erläuterungen:

Die Farbquelle gibt, wie der Name schon sagt, die Quelle an, aus der die zu benutzende Farbe stammt. Die Farbquelle wird bei dem COLOR-Befehl genauer erklärt.

Es gibt im Commodore-BASIC sogenannte Steuerzeichen. Das sind Zeichen, die nicht gedruckt werden, sondern irgend eine Funktion auslösen. Diese Zeichen, deren Funktion im Anhang übrigens aufgelistet sind, funktionieren im Grafikmodus nicht!

Oftmals reicht es nicht aus, Buchstaben nur in 8 Pixelschritten zu positionieren. Um nun auch Zeichen an allen Grafikkoordinaten auszudrucken, dient das folgende Programm, das die wichtigsten Zeichen zunächst auf den Bildschrim schreibt, und schließlich jeweils einen Buchstaben durch den SSHAPE-Befehl einer Variablen zuweist. Sie können diese Unterroutine in Ihre eigenen Programme einbauen, um die Buchstaben wirklich an jede Position im Grafikkoordinatensystem zu setzen.

```
 10 rem **********************************
 20 rem ***                            ***
 30 rem ***      program: char         ***
 40 rem ***      -------------         ***
 50 rem ***                            ***
 60 rem **********************************
 70 :
 80 rem *** schreiben des hauptzeichensatzes ***
 90 :
100 dim z$(191)              : rem zeichenarray vorbereiten
110 graphic 1,1              : rem grafik einschalten
120 for y=0 to 4
130 ::for x=0 to 39
140 ::::char ,x,y,chr$(64+z) : rem ein zeichen setzen
150 ::::z=z+1
160 ::::if z=128 then z=160  : rem bereichsüberschreitung
170 ::::if z>191 then 200    : rem testen
180 next x,y
190 :
200 z=0:for y=0 to 24 step 8
210 for x=0 to 319 step 8
220 :::sshape z$(z),x,y,x+7,y+7: rem zeichen in variable
230 :::box 1,x,y,x+7,y+7,,1  : rem uebertragen und loeschen
240 :::z=z+1
250 next x,y
```

Der COLOR-Befehl

Format : COLOR fq,fc

Parameter : - fq : Farbquelle.
0=Hintergrund (40-Zeichen-Anzeige)
1=grafischer Vordergrund
2=grafischer Mehrfarbenmodus 1
3=grafischer Mehrfarbenmodus 2
4=Rand
5=Textfarbe
6=Hintergrund (80-Zeichen-Anzeige)

- fc : Farbcode.
Folgende Farbcodes kann man wählen:

1=schwarz	9=hellbraun
2=weiß	10=braun
3=rot	11=rosa
4=grün	12=dunkelgrau
5=violet	13=grau
6=dunkelgrün	14=hellgrün
7=blau	15=hellblau
8=gelb	16=hellgrau

Beispiel : COLOR 4,8

Funktion : Definiert für jede mögliche Farbquelle eine Farbe.

Erläuterungen:

COLOR 0,fc ändert die Hintergrundfarbe des Bildschirms nur mit anschließendem GRAPHIC-Befehl. Das bedeutet, daß dieser Befehl nicht gleich die sichtbare Hintergrundfarbe ändert.
Wie schon bei der Beschreibung von anderen Befehlen angedeutet, möchte ich denen, die nichts mit den Farbquellen anfangen können, diese etwas näher bringen. Zunächst einmal dienen die Farbquellen 0-1 bzw. 4-6 dazu, ganz bestimmte Bildschirmstellen farbig zu gestalten. Hierzu folgt im Anschluß an die Definition der GRAPHIC-Befehl (siehe dort). Neben dieser Aufgabe haben die Farbquellen auch noch eine andere, sehr wichtige. Man kann diese Farbquellen bei allen Befehlen, die etwas zeichnen, abrufen und deren Inhalt als Zeichenfarbe benutzen. So ist es möglich, die Zeichenfarbe in Übereinstimmung mit der anderen, bereits benutzten, zu wählen.

Bei Ausgabe auf den RGB-Monitor, also die 80-Zeichen-Grafik, erhalten die einzelnen Farbcodes aus Hardware-Gründen eine andere Bedeutung:

```
1 = schwarz          9 = violett
2 = weiß            10 = braun
3 = rot             11 = hellrot
4 = hell-violett    12 = mittel-grau
5 = mittel-blau     13 = dunkel-grau
6 = grün            14 = hell-grün
7 = blau            15 = hell-blau
8 = gelb            16 = hell-grau
```

Der DRAW-Befehl

Format : DRAW fq,x1,y1 TO x2,y2 ....

Parameter : - fq : Farbquelle (siehe BOX)

: - x1,y1 : Koordinaten des Startpunktes

: - x2,y2 : Koordinaten des Endpunktes

Beispiel : DRAW 1,10,10 TO 20,50

Funktion : Zeichnen einer Linie

Erläuterungen:

Mit dem DRAW-Befehl können beliebig viele Linien aneinandergefügt werden. Beispiel: DRAW 1,10,10 TO 20,50 TO 30,5. Sind Start- und Endpunkte gleich, so wird ein Punkt gezeichnet. Dieser Effekt wird auch erzielt, wenn Sie nur die ersten 3 Parameter, ohne "TO", angeben.

Wird das "TO" durch ein Komma ersetzt, so werden jeweils nur Punkte an die entsprechenden Koordinaten gesetzt und nicht mit Linien verbunden.

Weiterhin haben Sie die Möglichkeit, vom Graphic-Cursor (siehe LOCATE) bis zu einem Endpunkt eine Linie zu ziehen. In diesem Fall müssen die Anfangskoordinaten entfallen.
Damit existieren eine Reihe verschiedener Möglichkeiten, diesen Befehl anzuwenden.

```
DRAW M,X1,Y1 TO X2,Y2    : Löschen oder Setzen einer Linie
DRAW M,X1,Y1 (, X2,Y2..): Setzen eines oder mehrerer Punkte
DRAW M       TO X1,Y1    : Zeichnen einer Linie vom
                           Grafikcursor noch x1,y1
```

```
100 rem *******************************
110 rem ***                         ***
120 rem ***        draw-demo        ***
130 rem ***        ---------        ***
140 rem ***                         ***
150 rem *******************************
160 :
170 graphic 1,1                    : rem grafik einschalten
180 for x=0 to 319 step 2
190 ::y=40 * sin(x / 25) + 100     : rem sinuswert errechnen
200 ::draw ,0,0 to x,y             : linie zeichnen
210 next x
220 :
230 graphic 0,1                    : rem grafik ausschalten
240 print "schoen, oder ?"
```

Der GRAPHIC-Befehl

Format : GRAPHIC mo,ls,tz
GRAPHIC CLR

Parameter : - mo : Modus. Folgende 6 Modi sind möglich:
0=Text mit 40 Zeichen / Zeile
1=hochauflösende Grafik (320*200 Punkte)
2=hochauflösende Grafik mit Text
3=Multicolorgrafik (160*200 Punkte)
4=Multicolorgrafik mit Text
5=Text mit 80 Zeichen / Zeile

- ls : Bestimmt, ob bei Aufruf der Modi 1-4 der Bildschirm gelöscht werden soll (1) oder nicht (0).

- ty : Legt bei den Modi 2 und 4 fest, ab welcher Zeile der Text beginnen soll.
Voreingestellt ist der Wert 19.

- GRAPHIC CLR : Gibt den reservierten Speicherplatz wieder frei.

Beispiel : GRAPHIC 1,1

Funktion : Aktiviert Grafikmodus und reserviert Speicherplatz (bzw. gibt ihn wieder frei)

Erläuterungen:

Bei den Modi 1 - 4 wird ein 9-KByte-Grafikspeicherplatz, der für die Darstellung der Grafik notwendig ist, am Anfang des BASIC-Programmspeichers reserviert und wird erst wieder durch GRAPHIC CLR freigegeben. Trotzdem müßten die verbleibenden 48 Kilobyte noch für eine sinnvolle Programmierung reichen !?
Weiterhin wird der Grafikmodus aktiviert. Außerdem werden alle Farben, die mit dem COLOR-Befehl neu definiert wurden, sichtbar.

Der LOCATE-Befehl

```
Format        : LOCATE x,y

Parameter     : - x,y  : Koordinaten des neuen Punktes.
                         (scaliert)

Beispiel      : LOCATE 100,130

Funktion      : Positioniert den grafischen Cursor auf
                dem Bildschirm.
```

Erläuterungen:

Der grafische Cursor ist nicht sichtbar und für alle grafischen Zeichenanweisungen der Ausgangspunkt. Nach der Ausführung einer Zeichenanweisung ist der letzte gezeichnete Punkt der neue Ausgangspunkt.

Im Klartext: Um eine bestimmte Figur mit Grafikbefehlen vom Grafikcursor aus zu zeichnen, müssen Sie einfach die ersten X- und Y-Koordinaten weglassen.

Der PAINT-Befehl

Format : PAINT fq,x,y,mo

Parameter : - fq : Farbquelle (siehe BOX)

: - x,y : Koordinaten des Startpunktes (scaliert)

: - mo : Legt fest, ob der auszumalende Bereich von der gewählten Farbe (0) oder einer anderen als der Hintergrundfarbe umgeben ist (1) (siehe Erläuterungen)

Beispiel : PAINT 1,100,100,0

Funktion : Füllen eines grafischen Bereiches mit : einer wählbaren Farbe.

Erläuterungen:

Der PAINT-Befehl dient dem Ausfüllen beliebiger Flächen. Diese müssen eindeutig begrenzt sein. Das bedeutet, daß die Begrenzung lückenlos sein muß. Logischerweise muß der Startpunkt innerhalb der auszufüllenden Fläche sein. Nach soviel "muß" kann man nun auch wählen: Die Farbquellen sind die gleichen, wie bei dem BOX-Befehl und werden natürlich auch mit dem COLOR-Befehl definiert (siehe dort). Nun gibt es aber noch zwei Möglichkeiten, wie die Fläche begrenzt ist: Die gleiche Farbe, wie die in der angegebenden Farbquelle, sowie eine fremde. Diese fremde Farbe kann nicht die Hintergrundfarbe sein, da die Begrenzung in einem solchen Fall nicht gegeben ist. Haben wir uns nun für eine Begrenzungsfarbe entschlossen, so

geben wir den dazugehörenden Modus mit dem PAINT-Befehl an: 0 für die gleiche Farbe (wie in der angegebenden Farbquelle), sowie 1 für die fremde Farbe.

Die RCLR-Funktion

```
Format      : x=RCLR(fq)

Parameter   : fq  : Farbquelle
                    0=Hintergrund (40-Zeichen-Anzeige)
                    1=Vordergrund
                    2=Multicolor 1
                    3=Multicolor 2
                    4=Rand
                    5=Text
                    6=Hintergrund (80-Zeichen-Anzeige)

Beispiel    : x=RCLR(1)

Funktion    : Liefert den aktuellen Farbcode der angege-
              benen Farbquelle.
```

Erläuterungen:

Dieser Befehl ermöglicht es uns, die Inhalte der Farbquellen abzufragen. Das ist dann nützlich, wenn in einem Programm Bedingungen an diese geknüpft werden. Dadurch ist es möglich, wenn nötig, Änderungen vorzunehmen. Nach der Ausführung dieses Befehls trägt die Variable x den abgefragten Inhalt.

Die RDOT-Funktion

Format : x=RDOT(n)

Parameter : - n : Gibt an, welcher Wert mit der Funktion gelesen werden soll:
o=x-Position des grafischen Cursors.
1=y-Position des grafischen Cursors.
2=Farbquell-Code, aus dem die Farbe des Punktes bei der aktuellen Position des grafischen Cursors resultiert.

Beispiel : x=RDOT(0)

Funktion : Liefert die aktuelle Position des grafischen Cursors oder den für die Position gültigen Farbcode der Farbquelle.

Der SCALE-Befehl

```
Format      : SCALE mo,xmax,ymax

Parameter   : - mo   : Modus.
                       0=Scalierung ausschalten
                       1=Scalierung einschalten

            : - xmax : Maximale x-Koordinate
                       HGR: 320-1023   MC: 160-1023

            : - ymax : Maximale y-Koordinate
                       HGR: 199-1023   MC: 199-1023

Beispiele   : SCALE 1,500,500
            : SCALE 0

Funktion    : Schaltet die Scalierung aus oder ein.
```

Bemerkungen:

Der SCALE-Befehl vermag den Wertebereich des Bildschirms in der Grafik zu erweitern. Dabei wird selbstverständlich die maximale Auflösung von 320*200 Punkten in der HGR und 160*200 in Multicolor beibehalten, und nur die eigentliche Koordinate mit einem bestimmten Faktor multipliziert. Dadurch werden alle Figuren auf dem Bildschirm logischerweise kleiner. Einen wesentlichen Vorteil hat dieser Befehl jedoch: Die lästige Umrechnung der Koordinaten bei einem erweiterten Koordinatensystem (dieses ist manchmal erforderlich, z. B. in der Statistik) entfällt. Beim Ausschalten der Scalierung ist nur der erste Parameter erforderlich.
Achtung! Sprite-Bewegungen werden entgegen des Commodore Handbuches nicht scaliert !

Der WIDTH-Befehl

Format : WIDTH n

Parameter :- n :1=einfache Strichstärke
2=doppelte Strichstärke

Beispiele : WIDTH 1
: WIDTH 2

Funktion : Setzt die Strichstärke für die Zeichenanweisungen fest. (CIRCLE,DRAW ect.)

Beispielprogramm:

```
 10 rem **********************************
 20 rem ***                            ***
 30 rem ***        width-demo          ***
 40 rem ***        ----------          ***
 50 rem ***                            ***
 60 rem **********************************
 70 :
 80 color 0,1                 :rem hintergrundfarbe
 90 color 4,1                 :rem rahmenfarbe
100 color 1,3                 :rem punktfarbe
110 graphic 1,1               :rem grafikseite einschalten
120 r=90                      :rem radius des kreises
130 w=1                       :rem strichstaerke
140 width w                   :rem strichstaerke setzen
150 circle 1,100,100,r        :rem kreis zeichnen
160 if w=1 then w=2:goto 140 :rem aendern der strichstaerke
```

```
170 w=1                    :rem aendern der strichstaerke
180 r=r-5                  :rem radius verkleinern
190 if r=5 then 170        :rem ende der grafik
200 goto 140
210 goto 210
```

Erläuterungen:

Wenn im Grafikspeicher nur ein einzelner Punkt gesetzt ist, so ist es aus hardwaremäßigen Gründen oft nicht möglich, die Farbe dieses Punktes genau zu bestimmen. Durch den WIDTH-Befehl kann das aber nun, da ja immer zwei Punkte nebeneinander gesetzt werden, nicht mehr passieren. Ein weiterer Vorteil, den der WIDTH-Befehl bietet, ist, daß nun auch HIRES-Bilder den MULTI-COLOR-Bilder ähnlicher werden, da ja jetzt auch nur noch eine Auflösung von 159 * 199 Punkten vorhanden ist.

1.3.2 Die Textgrafikbefehle

Die Textgrafik ist eigentlich nichts anderes, als die Gestaltung des Bildschirms mit normalen Text-Zeichen. Durch die nun folgenden Befehle ist es uns möglich, den Bildschirmaufbau sinnvoller als nur durch POKE oder PRINT Befehle zu gestalten.

Der PRINT-USING-Befehl

Format : PRINT #fn USING v$;au(;)

Parameter : - fn : Nummer der Ausgabedatei
1-127:Carriage return
128-255:Angefügter Zeilenvorschub
(Linefeed)
Für die normale Bildschirmausgabe
ist dieser Parameter nicht nötig.

: - v$: Definiert das Ausgabeformat

: - au : Ausdrücke, die formatiert ausgegeben werden sollen. Wenn mehrere
Ausdrücke behandelt werden sollen,
so werden sie mit Kommata oder
Semikolon getrennt. Hierbei ist zu
beachten, daß ein Komma keine Tabulierung bewirkt, wie dies beim
normalen PRINT-Befehl der Fall
ist.

Beispiel : PRINT USING "#,###.##";a;b;c

Funktion : Formatierte Ausgabe eines oder mehrerer
Ausdrücke.

Erläuterungen:

Die Definition des Ausgabebildes erfolgt mit Hilfe verschiedener Zeichen, die man beherrschen sollte:

String-Ausdrücke:

#: Dieses Zeichen steht für ein Zeichen im Ausgabebild. Hat der zu behandelnde Ausdruck mehr Zeichen als diejenigen Stellen, die mit # definiert sind, so werden die überzähligen Zeichen bei der Ausgabe nicht berücksichtigt. Ansonsten erfolgt die Ausgabe linksbündig.

=: Die Ausgabe erfolgt zentriert.

>: Die Ausgabe erfolgt rechtsbündig.

Numerische Ausdrücke:

#: Zeichen des Ausgabebildes. Die Ausgabe erfolgt rechtsbündig. Nicht benötigte Stellen werden mit Leerzeichen ausgefüllt. Wenn weniger #-Zeichen als Stellen im eingegebenen Ausdruck angegeben werden, so wird das Ausgabebild mit *-Zeichen gestaltet. (siehe untenstehendes Beispiel).

+,-: Eines dieser Zeichen bewirkt eine Vorzeichenangabe bei der Ausgabe. Die Zeichen können entweder am Anfang oder Ende der Ausgabedefinition stehen (siehe untenstehendes Beispiel).

.: Dieser Punkt legt die Dezimalpositionen fest. Sind weniger Stellen als in der angebenen Zahl vorhanden, so wird die Zahl gerundet. Dieses Zeichen darf außerdem nur einmal in der Definition verwendet werden (siehe untenstehendes Beispiel).

,: Dient zur besseren Lesbarkeit von größeren Zahlen. Im Ausgabebild erscheint ein Komma, was nicht mit dem Dezimalkomma verwechselt werden darf. Außerdem muß vor dem ,-Zeichen in der Definition ein #-Zeichen stehen (siehe untenstehendes Beispiel).

$: Vor der ersten Zahl im Ausgabebild wird ein Dollar-Zeichen gesetzt. Auch hier muß vor diesem Zeichen in der Definition ein #-Zeichen stehen. Wie aus dem PUDEF-Befehl ersichtlich wird, bedeutet dieses Zeichen, daß es möglich ist, der formatierten

Ausgabe ein beliebiges Zeichen voranzustellen. Ist dieses Zeichen noch nicht umdefiniert, so benutzt der Computer das Dollar-Zeichen (siehe untenstehendes Beispiel).

^^^^: Erwirkt eine Ausgabe im Exponentialformat. Nach diesen vier Zeichen muß noch ein + oder - in der Definition folgen. Das Exponentialformat ist, wie mathematisch versierte Leser sicher wissen, eine besondere Schreibweise von Zahlen. So wird der Dezimalpunkt hinter das erste Zeichen gesetzt. Die Anzahl der Stellen, die hinter der ersten Zahl existiert, wird extra ausgedrückt. Hierdurch wird die Schreibweise mancher Zahlen wesentlich kürzer.

Beispiel:

```
Normalformat          Exponentialformat
100000000000          1.0 E + 11
                oder    1 E + 11
```

Doch auch kleinere Zahlen lassen sich so ausdrücken. Im Gegensatz zu der geraden benutzten Schreibweise, wird das + durch ein - ersetzt. Wie Sie sicher schon festgestellt haben, wird bei der Benutzung des + der Dezimalpunkt nach rechts verschoben, um die alte Schreibweise zu erlangen. Jetzt ist es umgekehrt:

```
Normalformat          Exponentialformat
0.00000000001         1.0 E - 11
                oder    1 E - 11
```

Zur besseren Verständlichkeit nun noch ein Beispiel:

```
10 X=12345.12345
20 V$="+#$##,###,###.###"
30 FOR I=1 TO 10
40 PRINT USING V$;X
50 V$=LEFT$(V$,LEN(V$)-1)
60 NEXT I
```

Der PUDEF-Befehl

Format : PUDEF v$

Parameter : - v$: Ein Stringausdruck, mit der Länge von höchstens 4 Zeichen. Jede Position stellt ein bestimmtes Steuerzeichen im PRINT USING-Befehl dar. Dabei gilt:
Pos 1:Füllzeichen
Pos 2:Komma
Pos 3:Dezimalpunkt
Pos 4:Dollarzeichen

Funktion : Definiert bis zu 4 Steuerzeichen im PRINT USING-Befehl neu.

Erläuterungen:

Als Füllzeichen ist das Leerzeichen voreingestellt. Sonst jeweils das angegebene. Also wird der Dezimalpunkt auch als Punkt ausgegeben.

Beispiel : PUDEF " ., "
Funktion : Stellt die englische Schreibweise von Zahlen auf die deutsche um. (Vertauschen von Punkt und Komma). Achtung: Im Bedienungshandbuch von Commodore hat sich in diesem Beispiel der Fehlerteufel eingeschlichen; denn dort fehlt das erste Leerzeichen im Stringausdruck.

Beispiel : PUDEF "-   "
Funktion : Das Schriftbild wird von links mit "-"-Zeichen anstatt mit Leerzeichen aufgefüllt.

Die RWINDOW-Funktion

Format : x=RWINDOW(n)

Parameter : - n : 0=Liefert die letzte Zeilennummer des aktuellen Windows (0-24)
1=Liefert die letzte Spaltennummer des aktuellen Windows (0-79).
2=Liefert die eingestellte Bildschirmbreite (40 oder 80).

Funktion : Diese Funktion liefert die Parameter des aktuellen Windows.

Der WINDOW-Befehl

Format : WINDOW xl,yl,xr,yr,l

Parameter : - xl,yl : Koordinaten des linken, oberen Eckpunktes des Bildschirmfensters.

: - xr,yr : Koordinaten des rechten, unteren Eckpunktes des Bildschirmfensters.

: - l : Bewirkt, daß das Window nach der Definition gelöscht werden soll (1), oder nicht (0). Hier ist der Wert 0 voreingestellt.

Beispiel : WINDOW 10,10,30,30,1

Funktion : Definiert ein Window (Bildschirmfenster).

Beispielprogramm:

In diesem Window-Beispielprogramm, haben wir bewußt keine Erklärungen eingefügt. Starten Sie doch einmal das Programm und versuchen Sie, nach dem Sie Ihren Begeisterungsausbruch überwunden haben, dieses Programm zu verstehen.

```
100 rem ****************************************************
110 rem ***                                              ***
120 rem ***               window demo 128                ***
130 rem ***                                              ***
140 rem ****************************************************
150 :
160 scnclr : m=1
```

```
170 graphic 5
180 if m=1 then slow : graphic 0 : else : fast : graphic 5
190 color 0,1 : color 1,5 : color 4,1
200 scnclr
210 print ""tab(9*m)"w i n d o w  s a m p l e"
220 :
230 space$="                                        "
240 :
250 for z=1 to 11
260 :read x1,y1,x2,y2
270 :char ,x1*m-1,y1-1,"U"
280 :for zz=x1*m to x2*m
290 ::print "C";
300 :next zz
310 :print "I"
320 :for zz=y1 to y2
330 ::print tab(x1*m-1)" "tab(x2*m+1)" "
340 :next zz
350 print tab(x1*m-1)"J";
360 :for zz=x1*m to x2*m
370 ::print "C";
380 :next zz
390 print "K"
400 :window x1*m,y1,x2*m,y2:print chr$(27)+"@"
410 :print "program list :"
420 :sleep 1
430 :list
440 print chr$(19)+chr$(19)
450 next : restore : goto250
460 :
470 rem *** fenster datas ***
480 :
490 data 1,4,37,22
500 data 15,5,30,11
510 data 4,16,20,19
520 data 11,9,34,15
530 data 10,10,18,13
540 data 16,10,22,16
```

```
550 data 1,4,10,10
560 data 20,6,36,12
570 data 12,5,24,15
580 data 1,1,37,22
590 data 11,14,12,14
```

Der CHAR-Befehl (Textgrafik)

```
Format       : CHAR fq,x,y,z$,iv

Parameter    : - fq   : Farbquelle
                        0=Hintergrundfarbe
                        1=Vordergrundfarbe
                        2=Zusatzfarbe 1
                        3=Zusatzfarbe 2

               - x,y  : Koordinaten des Textanfangs.
                        x:0-39 bzw. 79
                        y:0-24

               - z$   : Text, der ausgegeben werden soll.

               - iv   : 0=Normale Darstellung
                        1=Inverse Darstellung

Beispiel     : CHAR 1,3,20,"Der CHAR-Befehl"

Funktion     : Plaziert einen bestimmten Text an einer
               bestimmten Bildschirmstelle.
```

Erläuterungen:

Im Text enthaltene Steuerzeichen werden hier im Textmodus ausgeführt, jedoch können nur Strings, keine Fließkommata oder Integervariablen durch den CHAR-Befehl positioniert werden. Es gibt jedoch einen kleinen Trick, um dieses zu erreichen. Wandeln Sie zunächst Ihre Zahl mit dem STR$-Befehl in einen String um und positionieren Sie diesen dann durch den Char-Befehl.

1.3.3 Sprite/Shape-Grafik

Die folgenden Befehle befassen sich mit der Sprite- bzw. Shapegrafik. Sprites sind selbst kleine Grafiken, die unabhängig voneinander bewegt werden können.

Diese Eigenschaft ist sicher nicht uninteressant für die Programmierung von Spielen, und wirklich interessante Spiele kommen ohne diese Technik nicht aus. Der C-128 ist in der Lage, 8 Sprites unabhängig voneinander zu verwalten. Diese Verwaltung können Sie mit Hilfe der nun beschriebenen Befehle steuern:

Die BUMP-Funktion

```
Format        : x=BUMP(n)

Parameter     : - n     :Gibt an, nach welchem Kriterium
                         die Funktion arbeiten soll:
                         1=Kollision mit einem Sprite
                         2=Kollision mit einer Anzeige
```

```
Funktion        : Die Funktion liefert ein Byte, aus
                : deren Bitmuster sich das oder die Sprites
                : ermitteln lassen, die seit der letzten BUMP
                : Abfrage miteinander kollidiert sind. Um die
                : Nummer eines Sprites zu ermitteln, müssen
                : Sie folgenden Algorithmus verwenden:

       IF BUMP(n) AND 2^sn = 2^sn THEN Sprite kollidiert
```

Bemerkungen:

Zur Verwendung dieser Funktion muß die Bedingung nicht unbedingt mit dem COLLISION-Befehl aktiviert worden sein.

Nach der Abfrage wird die Information gelöscht. Es ist also immer ratsam, das Ergebnis einer Kollision zunächst in einer Variablen zwischenzuspeichern.

Der COLLISION-Befehl

```
Format          : COLLISION typ,ze

Parameter       : - typ  : Definiert das Ereignis, das zur
                           Unterbrechung führt.
                           1=Sprite-Sprite-Kollision
                           2=Sprite-Anzeigedaten-Kollision
                           3=Lichtstift-Aktivierung

                : - ze   : Die Zeilennummer einer gültigen
                           Programmzeile, zu der im Falle
                           einer Unterbrechung verzweigt
                           werden soll.
```

```
Beispiel        : COLLISION 1,1000


Funktion        : Führt einen Unterprogrammsprung in
                  eine andere Programmzeile aus,
                  falls das angegebene Ereignis eingetreten
                  ist.
```

Erläuterungen:

Dieser Befehl verzweigt ein Unterprogramm, wenn das definierte Ereignis eingetreten ist. Beispiel: Zwei Sprites kollidieren miteinander. Der COLLISION-Befehl findet in der Hauptsache seine Anwendung bei Spielen (außer Typ 3).

Die Verzweigung wird wie eine GOSUB-Anweisung ausgeführt. Daher ist es unbedingt erforderlich, das Unterprogramm, in das verzweigt werden soll, mit einer RETURN-Anweisung abzuschließen.
Wird die Zeilennummer weggelassen, so wird die Verzweigung für das bestimmte Ereignis inaktiviert.
Es können mehrere Untersuchungen gleichzeitig aktiviert werden, allerdings wird immer nur eine bearbeitet.
Die BUMP-Funktion kann dazu verwendet werden, das Sprite zu bestimmen, das seit der letzten Abfrage eine Collision verursacht hat. (siehe dort)
Bemerkungen:

Beispielprogramm:

```
100 rem ***********************************************
110 rem ***                                         ***
120 rem ***             collision-demo 1            ***
130 rem ***             ----------------            ***
140 rem ***                                         ***
150 rem ***********************************************
160 :
170 color 0,1                    : rem rahmenfarbe
```

```
180 color 4,1                    : rem hintergrundfarbe
190 graphic 0
200 scnclr
210 :
220 for x=0 to 62                : rem sprite einlesen
230 :read g$
240 :sp$=sp$+chr$(dec(g$))
250 next
260 :
270 collision 1,390
280 :
290 for x=1 to 8
300 :sprite x,1,x+1
310 :movspr x,int(rnd(1)*255)#7
320 :sprsav sp$,x
330 next x
340 :
350 print "und alles irq-gesteuert";
360 get a$:if a$<>"" then 460
370 goto350
380 :
390 :rem collision unterprogramm
400 for z=1 to 2:color 4,z:next
410 color 4,1
420 return
430 :
440 rem 63 sprite-daten
450 :
460 rem ***********************************************
470 rem ***                                         ***
480 rem ***             collision-demo 2            ***
490 rem ***            -----------------            ***
500 rem ***                                         ***
510 rem ***********************************************
520 :
530 collision 1
540 sprite 4,0:sprite 5,0:sprite 6,0:sprite 7,0:sprite 8,0
550 scnclr
560 for x=1 to 3
570 ::r(x)=int(rnd(1)*360)
```

```
580 ::movspr x,r(x)#5
590 next x
600 :
610 for x=1 to 30
620 ::poke 1024+int(rnd(1)*999),160
630 next x
640 :
650 collision 1,680:collision 2,680
660 goto660
670 :
680 rem collision 2-unterprogramm
690 :
700 for x=0 to 3
710 :if bump(1)and2^x>0 then r(x+1)=360-r(x+1)
    :movspr x+1,r(x+1)#5
720 :if bump(2)and2^x>0 then r(x+1)=360-r(x+1)
    :movspr x+1,r(x+1)#5
730 next x
740 return
750 :
760 data 00 , 00 , 00 , 00 , 00 , 00 , 00
770 data 00 , 00 , 00 , 00 , 00 , 07 , 00
780 data 00 , 1f , c0 , 00 , 3f , e0 , 00
790 data 3f , e0 , 00 , 7f , f0 , 00 , 7f
800 data f0 , 00 , 7f , f0 , 00 , 3f , e0
810 data 00 , 3f , e0 , 00 , 1f , c0 , 00
820 data 07 , 00 , 00 , 00 , 00 , 00 , 00
830 data 00 , 00 , 00 , 00 , 00 , 00 , 00
840 data 00 , 00 , 00 , 00 , 00 , 00 , 00
```

Der GSHAPE-Befehl

Format : GSHAPE zk,x,y,mo

Parameter : - zk :Variable mit binärer Bildinformation (s. SSHAPE-Befehl)

: - x,z :Bildschirmkoordinaten der linken, oberen Ecke des darzustellenden Bildes (scaliert)

: - mo :Art der Bildschirmdarstellung:
0=Darstellung genau so, wie sie gespeichert wurden.
1=Invertierte Bildschirmausgabe
2=Überlagerung der Darstellung auf dem vohandenen Bild
3=Darstellung nur auf schon benutzten Bildpunkten
4=Schon benutzte Bildpunkte,auf die die neue Darstellung übertragen werden soll, werden invertiert.

Beispiel : GSHAPE z$,10,10,4

Funktion : Überträgt ein gespeichertes Shape auf
: den Grafik-Bildschirm.

Bemerkungen:

Die Stringvariable, die sich ergibt, ist voll kompatibel zu denen der Spritevariablen. Es ist also möglich, ein Sprite aus der Grafik herauszulsen, oder in eine solche hinein zu stempeln. Sinnvoll wäre dies vielleicht bei Layoutprogrammen oder ähnlichen Dingen.

Der MOVSPR-Befehl

```
Format      : MOVSPR n,x,y
              MOVSPR n,+/- x,+/- y
              MOVSPR n,wn#ge

Parameter   : - n    : Spritenummer (1-8)

              - x,y  : Koordinaten, bei denen das Sprite
                       positioniert werden soll.
                       (absolut)

              - +/- x,
                +/- y: Koordinaten, bei denen das Sprite
                       positioniert werden soll.
                       (relativ)

              - wn   : Winkel,der die Richtung angibt, in
                       der das Sprite bewegt werden soll
                       (0-360)

              - ge   : Bewegungsgeschwindigkeit des
                       Sprites (0-15)
```

Funktion : Erlaubt das Bewegen bzw. Positionieren eines Sprites.

Format 1 : MOVSPR n,x,y
Beispiel : MOVSPR 0,100,100
Funktion : Positioniert das Sprite bei (100/100).

Format 2 : MOVSPR n,+/- x,+/- y
Beispiel : MOVSPR 0,-30,-30
Funktion : Addiert die angegebenen Werte zu den momentanen Koordinaten des Sprites und positioniert es dort. Falls die alten Koordinaten (100/100) hießen, so wären die neuen (70/70). Bei dem Gebrauch von Variablen muß darauf geachtet werden, daß die Vorzeichen angegeben werden. Beispiel: MOVSPR 0,-x,-y bzw. MOVSPR 0,+x,+y

Format 3 : MOVSPR n,wn#ge
Beispiel : MOVSPR 0,90,8
Funktion : Bewegt das Sprite mit der Nummer 0 mit einer Geschwindigkeit von 8 auf dem Bildschirm von links nach rechts.

Beispielprogramm:

```
100 rem ***************************************************
110 rem ***                                             ***
120 rem ***               movspr-demo                   ***
130 rem ***              -------------                  ***
140 rem ***                                             ***
150 rem ***************************************************
160 :
170 color 0,1                   : rem rahmenfarbe
180 color 4,1                   : rem hintergrundfarbe
190 :
```

```
200 for x=0 to 62                  : rem sprite einlesen
210 :read g$
220 :sp$=sp$+chr$(dec(g$))
230 next
240 :
250 sprsav sp$,1
260 sprite      1,    1,    6,     0,     0,    0,     0
270 rem    nummer, ein,farbe, prior, xexp, yexp, modus
280 :
290 g  = 9.81                   : rem gravitation
300 ag = 3.21                   : rem gegenbeschleun.
310 lw = 0.04                   : rem luftwiderstand
320 sp = 5.00                   : rem geschwindigkeit
330 xk = 30                     : rem x-start
340 yk = 50                     : rem y-start
350 :
360 for s=1 to 120              : rem zeit 1/10 sekunden
370 :
380 :xk=xk+sp                   : rem geschwindigkeit horiz.
390 :sp=sp-lw                   : rem gegenbeschleunig. vet.
400 :if sp<0 then sp=0
410 :yk=yk+sy                   : rem geschwindigkeit vert.
420 :sy=sy+g/10                 : rem beschleunigung horiz.
430 :if yk>180 then sy=-sy/1.7: rem aufgeprallt
440 :movspr 1,xk,yk
450 next s
460 :
470 rem 63 sprite-daten
480 :
490 data 00 , 00 , 00 , 00 , 00 , 00 , 00
500 data 00 , 00 , 00 , 00 , 00 , 07 , 00
510 data 00 , 1f , c0 , 00 , 3f , e0 , 00
520 data 3f , e0 , 00 , 7f , f0 , 00 , 7f
530 data f0 , 00 , 7f , f0 , 00 , 3f , e0
540 data 00 , 3f , e0 , 00 , 1f , c0 , 00
550 data 07 , 00 , 00 , 00 , 00 , 00 , 00
560 data 00 , 00 , 00 , 00 , 00 , 00 , 00
570 data 00 , 00 , 00 , 00 , 00 , 00 , 00
```

Die RSPRITE-Funktion

```
Format       : x=RSPRITE(n,m)

Parameter    : - n    : Spritenummer (1-8)

             : - m    : 0=Liefert den Wert 1, wenn das
                          Sprite aktiviert ist, sonst 0
                        1=Liefert Spritefarbe
                        2=Liefert eine 0, wenn das Sprite
                          Priorität über den Hintergrund
                          hat, sonst 1
                        3=Liefert eine 1, wenn das Sprite
                          in x-Richtung gedehnt ist,sonst 0
                        4=Liefert eine 1, wenn das Sprite
                          in y=Richtung gedehnt ist,sonst 0
                        5=Liefert eine 1, wenn für das
                          Sprite der Mehrfarbenmodus aktiv
                          ist, sonst 0

Funktion     : Liefert die aktuellen Attribute eines
               Sprites.
```

Erläuterungen:

Sprites sind nicht nur dazu da, um sie, einmal in Bewegung gesetzt, zu vergessen. Es ist vielmehr sinnvoll, einige Informationen über die laufenden Sprites zu verarbeiten, um eine kontrollierte Bewegung auf dem Bildschirm zu erreichen. Diese Informationen liefert uns der RSPRITE-Befehl. Dadurch ist es uns möglich, eine sinnvolle Informationsverarbeitung zu betreiben.

Der SPRCOLOR-Befehl

```
Format          : SPRCOLOR z1,z2

Parameter       : - z1   : Setzt Zusatzfarbe 1 (1-16)

                : - z2   : Setzt Zusatzfarbe 2 (1-16)

Funktion        : Setzt für alle Sprites die Zusatzfarben
                  im Mehrfarbenmodus.
```

Erläuterungen:

Im Mehrfarbenmodus ist es uns möglich, Sprites mit Zusatzfarben zu versehen. Diese Zusatzfarben werden mit Hilfe des SPRCOLOR-Befehls definiert. Die Farbcodes (1-16) sind hierbei die gleichen, wie bei der HGR im 40-Zeichenmodus.

Der SSHAPE-Befehl

```
Format          : SSHAPE z$,x1,y1,x2,y2

Parameter       : - z$   : Zeichenkettenvariable, die den
                           gewählten Bildschirmbereich als
                           Binärinformation übernimmt.

                  - x1,y1: Koordinaten des linken, oberen
                           Eckpunktes des gewählten Bild-
                           schirmbereichs (scaliert)
```

- x2,y2: Koordinaten des rechten, unteren Eckpunktes des gewählten Bildschirmbereichs (scaliert). Voreingestellt ist die aktuelle Position des grafischen Cursors.

Beispiel : SSHAPE z$,10,10,20,20

Funktion : Übergibt den Inhalt des gewählten Bildschirmbereichs der Variable z$ in binärer Form.

Erläuterungen:

Der SSHAPE-Befehl ermöglicht es uns, den Inhalt eines Bildschirmbereichs zu "lesen" und in eine Variable zu übergeben. Hierdurch wird uns die Möglichkeit gegeben, bestimmte Grafiken zu verschieben, da ja der GSHAPE-Befehl die Umkehrung des SSHAPE-Befehls ist und den Inhalt einer Variable in einen Bildschirmbereich übergibt (siehe auch GSHAPE-Befehl). Es ist allerdings nicht möglich, einen beliebig großen Bildschirmbereich in eine Variable zu übergeben. Die Länge der zu übergebenen Zeichenkette beschränkt sich auf 255 Zeichen. Um Komplikationen mit der maximalen Länge zu vermeiden, kann man zwei Formeln benutzen, die die benötigte Länge berechnen:

Hochauflösend: l=INT((ABS(x1-x2)+1)/8+.99)*(ABS(y1-y2)+1)+4

Multicolor: l=INT((ABS(x1-x2)+1)/4+.99)*(ABS(y1-y2)+1)+4

Der Vorteil dieser Formeln liegt klar auf der Hand: Hat man sich einen Bildschirmbereich ausgesucht, so ist es vorteilhaft zu wissen, ob die Größe des Bereiches nicht die Möglichkeiten des Befehls übersteigt. Benutzt man die Formeln, so entgeht man den Fehlermeldungen des Computers.

Die RSPCOLOR-Funktion

Format : x=RSPCOLOR(n)

Parameter : - n : 1=Farbcode für die Sprite-Zusatzfarbe 1
: 2=Farbcode für die Sprite-Zusatzfarbe 2

Funktion : Liefert die aktuelle Zusatzfarbe für Sprites

Die RSPPOS-Funktion

Format : x=RSPPOS(n,m)

Parameter : - n : Nummer des zu untersuchenden Sprites (1-8)

: - m : 0=liefert aktuelle x-Koordinate
1=liefert aktuelle y-Koordinate
2=liefert Geschwindigkeit

Funktion : Liefert aktuelle Position bzw. Geschwindigkeit eines Sprites.

Sonstige Grafikbefehle

In diesem Kapitel werden Sie Befehle finden, die entweder bei keinem der Übergeordneten Kapitel unterzuordnen waren, oder bei denen es sich nicht gelohnt hätte, extra ein eigenes Kapitel zu schreiben. Betroffen hiervon sind im Grunde genommen nur zwei Befehle, die PEN-Funktion und die SCNCLR Anweisung.

Die PEN-Funktion

```
Format       : x=PEN(n)

Parameter    : - n    : 0=X-Koordinate der Lichtstift-
                          position auf dem grafischen
                          Bildschirm
                      : 1=y-Koordinate der Lichstift-
                          position auf dem Grafik-
                          bildschirm
                      : 2=Zeichen-Spaltenposition des
                          Lichtstiftes beim 80 Zei-
                          chenbildschirm
                      : 3=Zeichen-Zeilenposition des
                          Lichtstiftes beim 80 Zei-
                          chenbildschirm
                      : 4=Es wird 1 übergeben, falls
                          der Lichtstift seit der
                          lezten Abfrage aktiviert
                          wurde, anderfalls 0


Funktion              : Ermittelt den Zustand oder
                      : die Bildschirmkoordinaten des
                      : Lichtstiftes
```

Bemerkungen:

Die Koordinaten, die sich durch die PEN-Funktion ergeben, sind nicht scaliert. Bei Gebrauch des Lightpens sollten Sie eine möglichst helle Hintergrundfarbe auswählen, damit ein zuverlässiger Betrieb mit dem Lightpen gewährleistet ist.

Die SCNCLR-Anweisung

Format : SCNCLR(n)

Parameter

"n" ist ein ganzzahliger Wert mit folgender Bedeutung:

0 - Der 40-Zeichenbildschirm wird gelöscht.
1 - Der hochauflösende Grafikbildschirm wird gelöscht.
2 - Der hochauflösende geteilte Bildschirm wird gelöscht.
3 - Der Mehrfarben-Grafikbildschirm wird gelöscht.
4 - Der geteilte Mehrfarbengrafikbildschirm wird gelöscht.
5 - Der 80-Zeichenbildschirm wird gelöscht.

Bemerkung:

Wird "n" weggelassen, so wird der jeweils aktivierte Bildschirm gelöscht. Das Weglassen von "n" kann also entgegen des Commodore-Handbuches in allen Modi geschehen.

II. Der VIC II (8566)

2.1 Einführung zum VIC II

Der VIC II ist einer der beiden Grafikprozessoren im Commodore 128. Er ist für die Erzeugung eines Monitorbildes im 40-Zeichenmodus zuständig. Zusätzlich übernimmt er noch das gesamte Timing der dynamischen Rams, fragt die Zusatztastatur ab und organisiert die Umschaltung des Systemtaktes von 1 auf 2 Megahertz (und umgekehrt).

Die Ausgangssignale des VIC II liegen am Fernseherausgang (RF) sowie dem Composite-Videoausgang (VIDEO) an. Zwischen diesen beiden Ausgängen besteht folgender Unterschied:
Am RF-Ausgang liegt das modulierte Signal an, welches dann über das Antennenkabel zum Fernseher "gesendet" wird. Im Fernseher wird das Signal dann wieder demoduliert und in die drei Grundsignale Luminance (Leuchtdichte), Crominance (Farbart) und den Sound zerlegt. Aus diesen Signalen wird dann ein Bild komponiert und auf dem Fernseher dargestellt. Beim Composite-Video-Ausgang entfällt die Modulation und die Demodulation, d.h. die Signale werden direkt vom VIC II zum Monitor geführt und dort in Form eines Bildes ausgewertet.

Durch das Wegfallen der Modulation bzw. der Demodulation erhöht sich die Bildqualität wesentlich. Sollte ihr Fernseher also einen Video-Eingang besitzen, so sollten Sie diesen dem normalen Antenneneingang vorziehen.

Doch nun zu den Möglichkeiten, die uns der Videocontroller bietet. Es sind dies wie folgt:

- 40 Zeichen pro Zeile bei 25 Zeilen
- 16 Farben / normgerechtes PAL-Signal
- HI-RES Grafik (HGR/MC) mit 320*200 bzw 160*200 Punkten
- Multi-Color- und Extended-Color-Textmodus
- 8 Sprites und diverse Register zu deren Verwaltung

- Raster- und Spritekollisionsinterrupt
- Verschiebbarer Grafikspeicher, Video-RAM und Zeichensatz
- Verwaltung eines speziellen Farb-RAMs bzw. 16K RAM
- Taktauswahl/Anfrage der Zusatztastatur

2.2 Grundsätzliches zum VIC II

Unter diesem Kapitel "Grundsätzliches zum VIC II" wollen wir einen der wichtigsten Aspekte für die Programmierung des Videocontrollers besprechen.

Möglicherweise ist Ihnen schon aufgefallen, daß mit dem Register 24 des VIC's beispielsweise nur ausgewählt werden kann, ob der Grafikspeicher die oberen oder die unteren 8K des VIC-Adreßraums belegt, nicht aber festgelegt werden kann, wo dieser besagte "VIC-Adreßraum" nun eigentlich liegen soll. Aus diesem Grund wollen wir uns nun mit diesem "VIC-Adreßraum" eingehender befassen.

Der VIC besitzt nur einen 14 Bit breiten Adreßbus (A0-A13), von daher kann er nicht den vollen Adreßraum von 64K adressieren, wie der Prozessor, sondern eigenständig nur auf einen 16K Byte großen Speicher zugreifen.

Nun wäre es allerdings unpraktisch, ja sogar verschwenderischer Frevel, wenn man mit nur 16K für Grafikspeicher, Sprites, Text u.s.w. auskommen müßte, obwohl uns maximal 128K in der Grundversion zur Verfügung stehen. Wollte man beispielsweise drei Grafiken im Speicher haben, um diese nacheinander darstellen zu können, so müßte man jedesmal maximal 10K Ram für Grafikspeicher, Video-RAM und eventuell auch Farb-RAM, softwaremäßig kopieren.

Dieser Nachteil wurde auch schon bei der Entwicklung des Videocontrollers erkannt, so daß die fehlenden Adreßbits in den Bits 0 und 1 des Registers 0, der CIA II (56576/$DD00), zur

Verfügung gestellt wurden. Diese Bits werden nicht vom VIC oder der MMU kontrolliert, sondern müssen vom Programmierer gesetzt oder gelöscht werden.

Der Programmierer hat hierbei lediglich zu beachten, daß durch diese Adreßbits 14 und 15, die gesamten Basisadressen, von z.B. Video-RAM, Grafikspeicher, Spritepointer, Spritepuffer, (nicht aber vom Farb-RAM) um den, durch diese beiden Bits kodierten Wert, erhöht werden.

Würden Sie also in der Einschaltkonfiguration den VIC-Adreßraum nach 32768 ($8000) bis 49151 ($BFFF) verlegen, so läge der Bildschirmspeicher bei 33792 ($8400) und die Spritepointer von 34808 ($87F8) bis 34815 ($87FF). Der Farb-RAM wird hiervon jedoch nicht beeinflußt, da er vom VIC direkt adressiert wird und nicht verschiebbar ist (Siehe auch Kapitel 2.8).

Aufgrund der Tatsache, daß die Bits 0 und 1 des Registers 0, der CIA II, lowaktiv sind (0 gilt als 1 und umgekehrt), ergeben sich folgende Basisadressen für den VIC-Adreßraum:

Block	Bit 0	Bit 1	Wert durch Bit 0 kodiert	Wert durch Bit 1 kodiert	Anfang VIC II Adreßbereich	Ende VIC II Adreßbereich
0	1	1	= 0 +	0 =	0 ($0000) -	16384 ($3FFF)
1	1	0	= 0 +	16384 =	16384 ($4000) -	32767 ($7FFF)
2	0	1	= 32768 +	0 =	32768 ($8000) -	49151 ($BFFF)
3	0	0	= 32768 +	16384 =	49152 ($C000) -	65535 ($FFFF)

Nun hätten Sie aber keinen Commodore-Rechner, wenn nicht wieder irgendein Haken bei der Sache wäre. Und richtig, da das eigentliche Character-ROM bei 53248 ($D000) bis 51343 ($DFFF) liegt und nach dem Einschalten eigentlich für den Videocontroller nicht zu erreichen wäre, (Es ist der unterste 16K-Block ausgewählt und das Character-ROM liegt im vierten

Block.) existieren zwei Bereiche, die im 64er Modus einen Sonderstatus genießen. (Zum 128er Modus kommen wir gleich noch!) Diese beiden Bereiche liegen von 4096 ($1000) bis 8191 ($1FFF) und von 36864 ($9000) bis 40959 ($9FFF). Immer wenn der VIC versucht auf eine Adresse innerhalb dieser zwei Bereiche zuzugreifen, greift er auf das Character-ROM zu.

Leider hat diese Lösung aber auch eine Kehrseite: Es ist nicht mehr möglich, in diesen beiden Bereichen, im 64er Modus, Sprites oder den Grafikspeicher abzulegen, denn für den VIC liegt hier ja scheinbar das Character-ROM. Sie würden also Buchstaben und Grafikzeichen auf dem Bildschirm sehen, da ja der Aufbau von Grafikspeicher und Zeichensatz identisch ist.

Eine weitere Besonderheit, verursacht durch diese Sonderstellung der beiden Bereiche, macht sich darin bemerkbar, daß im Bereich von 53248 ($D000) bis 51343 ($DFFF), bei einem direkten Zugriff des VIC's, nur RAM vorhanden zu sein scheint. Wollen Sie also den Zeichengenerator ab 53248 ($D000) im RAM verändern, so müßen Sie ihn erst dorthin kopieren und dann aktivieren. Wie Sie dies machen und wohin Sie ihren neuen Zeichensatz am besten legen, erfahren Sie in den zugehörigen Kapiteln.

Sollten Sie jetzt glauben, das wäre alles gewesen, was es über die Speicherverwaltung des VIC's zu wissen gäbe, so haben Sie nur teilweise recht. Für den 64er Modus ist es hier genug der Einschränkungen, doch im 128er Modus kommt noch eine ganze Menge auf Sie zu. Als erstes wollen wir wieder auf das Zeropageregister 1 näher eingehen. Dieses besitzt nämlich im 128er Modus eine völlig andere Funktion. Ist Bit 2 dieser Speicherstelle gelöscht, so greift der Videocontroller im Bereich von 4096 ($1000) bis 8191 ($1FFF) und im Bereich von 36864 ($9000) bis 40959 ($9FFF), wie im 64er Modus, auf das Character-ROM zu. Es gelten daher die weiter oben beschriebenen Einschränkungen. Ist Bit 2 allerdings gesetzt, so greift der Videocontroller in allen Speicherbereichen grundsätzlich auf den RAM zu. Es ist dann auch möglich, in den beiden oben angegeben Bereichen den Grafikspeicher oder den Video-RAM abzulegen, wie dies vom Betriebssystem gemacht wird.

Außer dieser einen neuen Funktion besteht im 128er Modus noch folgend Besonderheit:
Wie Sie bereits wissen, können Sie zu den beiden serienmäßig eingebauten RAM-Bänken noch zwei weitere RAM-Bänke anschließen, doch laut der oben gegebenen Beschreibung ist es angeblich nur möglich, den VIC auf die RAM-Bank null zugreifen zu lassen. Um dieses Problem zu lösen, hat man zwei Bits in der MMU (Memory Management) dafür verwendet, zu bestimmen, aus welcher Bank der VIC seine Informationen beziehen soll. Dieses MMU-Register liegt an der Adresse 54534 ($D506) und bestimmt mit seinen obersten beiden Bits die ausgewählte RAM-Bank, wie aus der folgenden Tabelle ersichtlich:

```
Adresse 54534 ($D506)

Bit 7 ! Bit 6 ! Bank
--------------------
  0   !   0   ! 0
  0   !   1   ! 1
  1   !   0   ! 2 (0)
  1   !   1   ! 3 (1)
```

Die in Klammern angegebenen Banknummern stellen die ausgewählte Bank dar, die selektiert ist, wenn keine RAM-Erweiterung präsent ist.

Durch diese Möglichkeit der hardwaremäßigen Umschaltung zwischen mehreren RAM-Banken ergeben sich eine Vielzahl von Nutzungsmöglichkeiten. Als Beispiel wollen wir Ihnen hier die Umschaltung zwischen zwei verschiedenen Video-RAMs demonstrieren. Durch die rasche Umschaltung scheint das Bild nach unten wegzurollen, was natürlich nur auf einer optischen Täuschung beruht. Doch stellen Sie sich erst einmal die Nutzung dieser Umschaltung in einem Trickfilmgenerator vor. Sie könnten dann alle Bilder einladen und nacheinander, innerhalb von Sekundenbruchteilen anzeigen, wodurch ein Trickfilmeffekt erreicht würde. Sollte es Ihnen jetzt schon in den Fingern jucken, so lassen Sie ruhig Ihrem Programmiertrieb freien Lauf, denn

das einzige, was Sie bei der Programmierung zu beachten haben ist, daß das oben genannte MMU-Register bei einem RUN-STOP/RESTORE nicht zurückgesetzt wird, Sie dieses also von Hand erledigen müssen.

Programmlisting:

```
100 rem *************************
110 rem *   bank switch demo    *
120 rem *************************
130 :
140 hi=int(2048/256) : rem high-variablestart
150 lo=2048-hi*256 :   rem  low-variablestart
160 poke 47,lo : poke 48,hi : rem setzen
170 clr : rem alle variablenzeiger loeschen
180 :
190 for i=1024 to 2023
200 :   bank 0 : rem videoram i
210 :   poke i,64
220 :   bank 1 : rem videoram ii
230 :   poke i,111
240 next i
250 :
260 rem ******************************
270 rem *   vic-bank's umschalten    *
280 rem ******************************
290 :
300 bank 15 : i/o bereich auswaehlen
310 :
320 poke 54534,xor (peek (54534),64) : rem bit 6 umdrehen
330 :
340 for pa=1 to 50 : next pa : rem pause
350 :
360 goto 320 : rem und weiter
```

2.3 Registerbeschreibung des 8566

Der VIC II-Chip besitzt 49 Register, die hier in einer Kurzübersicht erklärt werden. Um die Adresse eines Registers zu ermitteln, müssen Sie die Registernummer zu der Basisadresse des Videocontrollers (53248 / $D000) addieren. Diese Adreßierung ist möglich, da die Register des VIC II direkt les- bzw. beschreibbar sind. Beim VDC (das ist der 80-Zeichencontroller!) ist dies nicht gegeben.

Register 00: X-Koordinate von Sprite 0
Dieses Register enthält die untersten acht Bits der X-Koordinate von Sprite 0.

Register 01: Y-Koordinate von Sprite 0
Wie oben, jedoch ist dieses Register für die Y-Koordinate zuständig.

Die nun folgenden 7 Registerpaare haben den gleichen Aufbau und die gleichen Funktionen, wie die ersten beiden Registerpaare.

Register 16: Most Significant Bits (MSBs) - Höherwertige Bits der X-Koordinate
In diesem Register ist für jedes Sprite vermerkt, ob die X-Koordinate größer als 255 ist. Soll das der Fall sein, muß das dem Sprite zugeordnete Bit gesetzt werden. Hierbei steht Bit 0 für Sprite 0, Bit 1 für Sprite 1 u.s.w.

Register 17: Erstes Steuerregister
Bit 0-2 Vertikales Scrolling in Rasterzeilen
Bit 3 24 Zeilen (0) / 25 Zeilen (1)
Bit 4 Bildschirm aus (0) / ein (1)
Bit 5 Textgrafik (0) / Einzelpunktgrafik (1)
Bit 6 Extended Color Mode aus (0) / ein (1)
Bit 7 8. Bit (MSB) von Register 18

Register 18: Rasterzeilensteuerung
In diesem Register können Sie die Bildschirmrasterzeile festlegen, bei deren Strahlendurchlauf ein Interrupt ausgelöst werden soll.
Bei einem Lesezugriff gibt dieses Register die Nummer der Rasterzeile an, die der Videocontroller gerade aufbaut. Das 8. Bit dieses Registers wird durch Bit 7 im Register 17 dargestellt.

Register 19: X-Koordinate des Lightpens nach einem Impuls.

Register 20: Wie oben, jedoch Y-Koordinate des Lightpens.

Register 21: Sprite an/aus
Wird das einem Sprite zugeordnete Bit in diesem Register gesetzt, so ist das Sprite aktiviert, andernfalls desaktiviert.

Register 22: Zweites Steuerregister
Bit 0-2 Horizontales Scrolling in Rasterpunkten
Bit 3 38 Spalten (0) / 40 Spalten (1)
Bit 4 Multicolormodus aus (0) / ein (1)
Bits 5-7 Keine Funktion

Register 23: Vertikale Spritevergrößerung
Alle Sprites, deren Bits in diesem Register gesetzt sind, werden in vertikaler Richtung expandiert.

Register 24: Basisadressen von Zeichengenerator,Video-RAM und Grafikspeicher

Bit 0 unbenutzt
Bit 1/2 Adreßbits 11 und 12 des Zeichen generators
Bit 3 Text: Adressenbit 13 des Zeichen-generators
Grafik: Auswahl zwischen den unteren (0) und den oberen (1) 8k des VIC-Adreßraums
Bit 4-7 Adreßbits 10-13 des Video-RAMs

Anmerkung:

Die Auswahl des vom VIC adressierbaren Speicherbereichs wird durch die untersten beiden Bits des Registers 0 der CIA 2 (56576) getroffen. Diese beiden Bits stellen die Adressenbits 14 und 15 der Anfangsadresse dar und sind low-aktiv. 'low-aktiv' bedeutet, daß ein gesetztes Bit als nicht gesetzt gilt und umgekehrt.
Um zum Beispiel die HI-RES Grafik an die Adresse 57344 ($E000) zu legen, müssen die ersten beiden Bits der Speicherstelle 56576 ($DD00) auf 0 und das erste Bit des VIC-Registers 24 auf 1 gesetzt werden (32768+16384+8192 = 57344).

Register 25: Interrupt Request Register (IRR) - Interrupt Rückfrage

Bit 0 Interruptauslöser ist Register 18 (Raster zeile)
Bit 1 Interruptauslöser ist Register 31 (Sprite-Hintergrund Kollision)

Bit 2 Interruptauslöser ist Register 30 (Sprite-Sprite Kollision)
Bit 3 Interruptauslöser ist der Lightpen
Bit 4-6 Diese Bits haben keine Funktion
Bit 7 Dieses Bit ist gesetzt, wenn eines der anderen Bits gesetzt ist.

Nach Auslösung eines Interrupts setzt der VIC dieses Register nicht selbständig zurück. Dieses können Sie aber einfach durch das Zurückschreiben des gerade gelesenen Wertes in das IRR erreichen.

Register 26: Interrupt Mask Register (IMR) - Interrupt Maske
Die Belegung dieses Registers entspricht der von Register 25.
Ist ein Bit im IMR und im IRR gesetzt, wird ein Interrupt (IRQ) ausgelöst.

Wollen Sie ein Bit im IMR setzen, so schreiben Sie dieses mit gesetztem 7. Bit in das IMR (128+2^Bitnummer). Hierdurch wird das gewünschte Bit gesetzt und alle anderen Bits in ihrem alten Zustand belassen. Das Löschen einzelner Bits erfolgt analog, nur daß das 7. Bit hierbei gelöscht sein muß.

Register 27: Hintergrundpriorität
Mit diesem Register können Sie bestimmen, ob ein Sprite hinter (0) oder vor (1) der Grafik liegen soll.

Register 28: Multicolor-Sprite
Jedem Sprite ist ein Bit zugeordnet; ist dieses gesetzt, wird das Sprite im Multicolor-Modus dargestellt.

Register 29: Horizontale Spritevergrößerung
Wie Register 23, nur für die horizontale Vergrößerung.

Register 30: Sprite-Sprite-Kollision
In diesem Register wird festgehalten, ob und welche Sprites kollidiert sind. Bei einer Kollision wird außer diesen Bits noch Bit 2 im IRR gesetzt.

Dieses Register müssen Sie durch Auslesen nach jeder Kollision wieder löschen, da der VIC das nicht für Sie erledigt.

Register 31: Sprite-Hintergrund-Kollision
Berührt ein Sprite einen Grafikpunkt, so wird das dem Sprite zugeordnete Bit und Bit 1 im IRR gesetzt.

Auch dieses Register müssen Sie wie Register 30 nach Auftreten der Kollision wieder zurücksetzen.

Register 32: Rahmenfarbe

Register 33: Hintergrundfarbe 0 (Vordergrundfarbe)

Register 34: Hintergrundfarbe 1, Multicolorfarbe 0 (Text)

Register 35: Hintergrundfarbe 2, Multicolorfarbe 1 (Text)

Register 36: Hintergrundfarbe 3

Register 37: Multicolorfarbe 0 (Sprites)

Register 38: Multicolorfarbe 1 (Sprites)

Register 39-46: Farben der Sprites 0-7

Register 47: Keyboard-Control-Register

- Bit 0-3 Diese Bits geben den Zustand der vier Interface-Pins der Zusatztastatur an
- Bit 4-7 Diese Bits haben keine Funktion und sind immer gesetzt.

Register 48: Taktauswahl

- Bit 0 Dieses Bit bestimmt, ob der System-takt ein (0) oder zwei (1) Megahertz betragen soll. Beträgt dieser zwei Mhz, so ist der VIC nicht mehr in der Lage, ordnungsgemäß zu arbeiten.
- Bit 1 Die genau Bedeutung dieses Bits ist leider noch unbekannt. Es scheint aber für die völlige Desaktivierung des VIC's verantwortlich zu sein und wird vom Betriebssystem nicht genutzt. Wird das Bit gesetzt, ändert sich auch die Synkronisationsfrequenz des Videocontrollers, wodurch das Bild anfängt zu flimmern.
- Bit 2-7 Diese Bits des Registers haben keine Funktion.

2.4 Die Betriebsarten

Genauso wie auf einem Fernseher verschiedene Programme dargestellt werden können, so kann man auch den VIC II in verschiedenen Betriebsarten betreiben. Grundsätzlich existieren folgende drei Hauptmodi, die noch weiter unterteilt sein können:

- Einzelpunktgrafik (HI-RES)
- Sprites
- Textgrafik

Zusätzlich lassen sich die verschiedenen Betriebsarten auch untereinander mischen. So ist es beispielsweise möglich, Sprites sowohl in der HI-RES-Grafik als auch im Textmodus darzustellen. Dies eröffnet dem trickreichen Programmierer eine Fülle an Kombinationsmöglichkeiten. Diese Funktionen für Sie nutzbar zu machen, ist nun das Ziel der nächsten Kapitel, in denen detailiert alle Einzelheiten der verschiedenen Grafikmodi besprochen und anhand anschaulicher Programme demonstriert werden.
Alle Angaben und Programme beziehen sich ausschließlich auf den 64er Modus, da der Basicinterpreter im 128er Modus die VIC-Register laufend korrigiert und Ihre POKEs daher maximal einen Interrupttakt (1/60 Sekunde) lang Wirkung hätten. Grundsätzlich sind natürlich alle Funktionen auch im 128er Modus nutzbar, allerdings nur über die komfortablen BASIC-Befehle oder durch Assemblerprogramme.

2.4.1 Die Verwaltung der HI-RES

Der VIC II-Chip kann in zwei verschiedenen Grafikmodi betriebenen werden. Es sind folgende:

- Normale Einzelpunktgrafik (HGR)

und die

- Multicolor Grafik (MC)

Der Unterschied zwischen beiden besteht darin, daß die HGR eine Auflösung von 320*200 Punkten bei zwei Farben pro 8*8 Punkteblock besitzt, wärend in der MC die X-Auflösung halbiert (160*200) und die Farbauflösung verdoppelt wird (4 Farben pro 4*8 Punkteblock).
Damit der VIC nun weiß, wie die Grafik aussehen soll, muß diese in dem sogenannten Grafikspeicher abgelegt werden. Da die maximale Auflösung 320*200 Punkte, also insgesamt 64000 Punkte beträgt, ist es notwendig, eine spezielle Kodierung der Punktmatrix zu schaffen. Denn würde jeder Punkt ein Byte be-

legen, hätte der Grafikspeicher schon einen Umfang von 64K. Da bei einem 64k großen Grafikspeicher kein Platz mehr für Basicprogramme oder den Farbspeicher bleiben würde, ist der Grafikspeicher folgendermaßen kodiert :
Jeweils acht Punkte werden von einem Byte festgelegt, was zur Folge hat, daß der Grafikspeicher nur noch ein Achtel der Größe hat, die er hätte, wenn jeder Punkt durch ein Byte dargestellt würde.

Nun wäre es durchaus logisch und praktikabel, wenn diese Bytes eines an dem anderen liegen würden, also etwa so:

```
    Byte 0           Byte 1
7 6 5 4 3 2 1 0 - 7 6 5 4 3 2 1 0 - 7 6 5  usw.
```

Leider ist dies nicht so, da diese Anordnung mit der Farbgebung recht schwierig zu koordinieren gewesen wäre. Praktikabler aber war die Anordnung von jeweils 8 Bytes in einem Block untereinander. Von diesen Blocks bilden dann 40 eine Zeile.
Diese Anordnung entspricht übrigens in etwa dem Aufbau des Zeichengenerators, in dem die Zeichen auch immer in 8-Byte-Blöcken abgelegt sind.

Aufgrund dieser Grafikverwaltung wird ein Punkt dann dadurch gesetzt, daß das korrespondierende Bit im Speicher auf 1 gesetzt wird. Das folgende Schaubild soll Ihnen noch einmal den Aufbau des Grafikspeichers bildlich demonstrieren:

```
                 Spalte 1          Spalte 2          Spalte 3
----------------------------------------------------------------
        ! Byte 0:........! Byte 0:........! Byte 0:........!
        ! Byte 1:........! Byte 1:........! Byte 1:........!
        ! Byte 2:........! Byte 2:........! Byte 2:........!
Zeile 1! Byte 3:........! Byte 3:........! Byte 3:......  !
        ! Byte 4:........! Byte 4:........! Byte 4:....    !
        ! Byte 5:........! Byte 5:........! Byte 5:...     !
        ! Byte 6:........! Byte 6:........! Byte 6:        !
        ! Byte 7:........! Byte 7:........! Byte 7:        !
----------------------------------------------------------------
        ! Byte 0:........! Byte 0:........
        ! Byte 1:........! Byte 1:........
        ! Byte 2:........! Byte 2:........
Zeile 2! Byte 3:........! Byte 3:......
        ! Byte 4:........! Byte 4:....
        ! Byte 5:........! Byte 5:...
        ! Byte 6:........! Byte 6:
        ! Byte 7:........! Byte 7:
----------------------------------
                 Spalte 1          Spalte 2
....
...
..
.
.
```

Um nun jeden Punkt genau zu definieren, teilt man das Grafikfenster in 25 Zeilen (0-24) mit jeweils 40 Spalten (0-39) auf. Jede dieser Spalten besteht für sich wieder aus 8 Bits (0-7), den sogenannten senkrechten Reihen. Jede Zeile wiederum besteht ebenfalls aus 8 übereinander angeordneten waagerechten Reihen. Der Grafikschirm kann also in 40*8 senkrechte Reihen (0-319) und 25*8 waagerechte Reihen (0-199) aufgeteilt werden. Dies ergibt eine Auflösung von 320*200 Punkten.

Um nun einen Grafikpunkt eindeutig zu bestimmen, könnte man sagen: Punkt X ist das Yte Bit im Byte Z. Diese Einteilung wäre allerdings kompliziert und unübersichtlich. Daher definiert man die Koordinaten eines Punktes wie folgt:

Die X-Koordinate wird mit der Nummer der senkrechten Reihe und die Y-Koordinate mit der Nummer der waagerechten Reihe gleichgesetzt. Die Koordinate 111/49 wäre also der Schnittpunkt der 111ten senkrechten Reihe mit der 49ten waagerechten Reihe. Oder noch genauer das 0te Bit des 13ten 8*8 Blocks in der 0ten waagerechten Reihe der 6ten Zeile.

Die Aufteilung des Bildschirms in 319 mögliche X-Koordinaten und 199 mögliche Y-Koordinaten entspricht übrigens in etwa dem Koordinatensystem der Mathematik. Hier jedoch liegt der Koordinatenursprung in der linken oberen, statt in der linken unteren Ecke. Somit werden dann die Y-Koordinaten von oben nach unten aufsteigend gezählt. Um also den Koordinatennullpunkt auch im Grafikspeicher nach unten links zu legen, müßte man die Y-Koordinate durch folgende Formel umrechnen:

Yn = Maximale Anzahl der waagerechten Reihen minus der Y-Koordinate

also: Yn = 199 - Y

Doch obriges nur für die Interessierten. Viel wichtiger für uns sollte sein, wie wir die X- und Y-Koordinaten wieder in das Format des Grafikspeichers umrechnen. Denn die Aufteilung in 319 X- und 199 Y-Koordinaten ist eine Vereinbarung unsererseits. Der Computer versteht diese Angaben nicht. Also müssen wir den Punkt wieder durch das Xte Bit im Yten Byte für den Computer verständlich angeben.

Um die Zeile (0-25), in der sich der Punkt befindet, zu ermitteln, müssen wir die Y-Koordinate lediglich ohne Rest durch 8 dividieren.

Zeile = INT(Y/8)

Der Vorkommateil des Quotienten Y/8 gibt dann die Zeile an, während über dem Nachkommateil die horizontale Reihe bestimmt werden kann. Da jede Zeile aus 40 nebeneinander liegenden Acht-Byte-Päckchen, also 320 Bytes, besteht, müssen wir die weiter oben ermittelte Y-Adresse mit 320 multiplizieren:

Relative Adresse der horizontale Reihe = INT(Y/8)*320

Die untersten drei Bits (Divisionsrest) des Y-Wertes geben nun die Reihe in der Zeile an und müssen auch noch zuaddiert werden. Unsere Formel zur Berechnung der horizontalen Reihe sieht daher jetzt so aus:

Adresse der horizontale Reihe = INT(Y/8)*320+(Y AND 7)

Nun wissen wir nur noch nicht, welches Bit in dem wievielten Byte der Reihe wir ansprechen müssen. Die Anzahl der Bytes gerechnet vom Reihenstart errechnet sich so:

Bytenummer = INT(X/8)*8 oder Bytenummer = X AND 504

Wir mußten zunächst durch 8 teilen, da ein Byte 8 Punkte darstellt. Dann jedoch multiplizieren wir das Ganze (ohne Rest) wieder mit 8, da jeweils 8 Bytes hintereinander einen 8*8 Block bilden. Durch die bis jetzt erhaltenen Formeln können wir wie folgt eindeutig die Adresse des Bytes ermitteln, in dem sich der gesuchte Punkt befindet:

Adr. = Grafikadr. + INT (Y/8) * 320+(Y AND 7) + X AND 504

Die Nummer des jeweiligen Bits errechnen wir, wie auch schon die horizontale Reihe, durch die 'AND' Funktion.

Also: Bitnummer = 7-(X AND 7)

(Rechnen Sie doch einmal nach!)

Diesen Wert benutzen wir jetzt als Exponent zur Basis 2, um in BASIC den Wert dieses Bits zu berechnen, den wir dann direkt als Maske zum Setzen eines Punktes verwenden können.
Wert = 2^(7-(X AND 7))

Jetzt kennen wir die beiden Formeln zur Errechnung der Adresse und der Maske eines Punktes. Setzen wir also nun einen Punkt in der HGR durch folgendes kleines Programm:

Programmlisting:

```
100 rem ********************
110 rem *   punkt setzen   *
120 rem ********************
130 :
140 dim a$(10000) : clr : rem grafikspeicher löschen
150 :
160 v = 53248 : rem basisadresse vic
170 :
180 print chr$ (147);
190 :
200 poke v+17,peek (v+17) or 32 : rem grafik an
210 poke v+24,24 : rem grafikspeicher bei 8192
220 :
230 for i=1024 to 2023 : poke i,7+0*16 : next : rem farbe setzen
235 :
240 rem **********************
250 rem *   punkt berechnen   *
260 rem **********************
265 :
270 x=111 : y=49 : bs=8192 : rem x/y koordinaten, basisadresse
280 :
290 ad=bs+int (y/8)*320+(y and 7)+int (x/8)*8
300 mk=2^(7-(x and 7))
310 :
320 poke ad,peek (ad) or mk : rem punkt hinzu 'or'en
330 :
```

```
340 wait 198,1 : poke 198,0
350 :
360 poke v+17,peek (v+17) and not 32 : rem grafik aus
370 poke v+24,21
380 :
390 end
```

2.4.1.1 Die Lage des Grafikspeichers

Um diesem Kapitel in angemessener Weise folgen zu können, sollten Sie das Kapitel 2.2 gelesen haben.

Wie Sie mittlerweile wissen, umfaßt der Grafikspeicher 8K RAM. In diesem RAM ist die Punktmatrix abgelegt, die auf dem Bildschirm dargestellt wird. Die Auswahl des Speicherbereichs, in dem der Grafikspeicher liegen soll, geschieht u. a. durch das Register 24 des Videocontrollers. Dieses Register ist allgemein für die Lage von Grafikspeicher, Video-RAM und Zeichensatz zuständig und wird uns noch öfter begegnen.

Für die Umschaltung des Grafikspeichers dient Bit 3, vorausgesetzt Bit 5 im VIC-Register 17 ist gesetzt, der Grafikmodus also eingeschaltet. Andernfalls dient es als Adreßbit 11 der Zeichenbasis, worauf wir allerdings jetzt noch nicht konkreter eingehen wollen. Falls Bit 1 des Registers gesetzt ist, ist auch Bit 14 in der Adresse des Grafikspeichers gesetzt. Konkreter heißt das: Ist das Bit gesetzt, muß zur Basisadresse des VIC-Adreßraums (#2.2) der Wert 8192 ($2000) hinzuaddiert werden, um die reale Startadresse des Grafikspeichers zu erhalten. Die Basisadresse des Grafikspeichers erhalten wir dann aus der Basisadresse des VIC-Adreßraums und dem Wert, der sich durch Bit 1 des VIC-Registers ergibt.

Setzen wir also in der Einschaltkonfiguration Bit 3 im Register 24, so liegt jetzt der Grafikspeicher bei 8192 ($2000). Probieren wir dies einmal aus:

```
10 v=53248 : rem basisadresse vic
20 poke v+17,peek (v+17) or  32 : rem grafik an
30 poke v+24,peek (v+24) or 2^3 : rem lage bei 8192
```

Nachdem Sie dieses Programm eingegeben und gestartet haben, liegt der Grafikspeicher bei 8192 ($2000). Dies wollen wir sofort ausprobieren, indem wir die Grafik löschen. Fügen Sie also die folgenden Zeilen dem oben stehenden Programm an und probieren Sie es aus.

```
40 for i=8192 to 8192+8000
50 poke i,0 : rem alle bits loeschen
60 next i
```

Haben Sie alles richtig eingegeben, so sollten Sie jetzt sehen, wie der Grafikspeicher langsam gelöscht wird. Übrigens bekommen Sie durch dieses kleine Programm noch einmal den Aufbau des Grafikspeichers demonstriert; denn, wenn Sie einmal genau hinschauen, erkennen Sie, daß immer erst 8 Bytes untereinander gelöscht werden und dann zum nächsten 8*8 Block gesprungen wird. Dies geschieht solange, bis alle 8000 Bytes auf Null gesetzt sind.

Zur Veranschaulichung haben wir Ihnen am Ende dieses Abschnitts noch eine Tabelle aller möglichen Grafikspeicherlagen in Abhängigkeit vom VIC-Adreßraum und Bit 3 des VIC-Registers 24 aufgelistet. Welche Funktion die CIA im Zusammenhang mit der Grafikspeicherlage hat, sollten Sie noch einmal im Kapitel #2.2 nachlesen, falls es Ihnen nicht mehr gegenwärtig ist!

Tabelle aller möglichen Grafikspeicherlagen:

```
CIA CIA VIC
 B | B | B |Höher-   |
 I | I | I |wertiges|
 T | T | T |Adreß-   |
 0 | 1 | 1 |byte     |
------------------------------------------------
 1   1   0  00000000      0 +     0 +    0 =     0 *
 1   1   1  00100000      0 +     0 + 8192 =  8192
 1   0   0  01000000      0 + 16384 +    0 = 16384
 1   0   1  01100000      0 + 16384 + 8192 = 24576
 0   1   0  10000000 32768 +     0 +    0 = 32768 *
 0   1   1  10100000 32768 +     0 + 8192 = 40960
 0   0   0  11000000 32768 + 16384 +    0 = 49152
 0   0   1  11100000 32768 + 16384 + 8192 = 57344
```

*) Siehe Ausnahmen im Kapitel 2.2.

2.4.1.2 Die Farbgebung in der HGR

Wie Ihnen vielleicht bekannt ist, kann der VIC Zeichen, Sprites oder Grafiken in 16 verschiedenen Farben darstellen. Bei 64000 Punkten benötigte man also minimal 32000 Bytes, wenn die Farbe eines jeden Punktes jeweils durch 4 Bits kodiert und jeder Punkt in seiner eigenen Farbe dargestellt werden sollte. Nun besitzen Sie aber einen Commodore 128 und keinen speziellen Grafikcomputer. Aus diesem Grund ist die Farbauflösung des VIC II relativ eingeschränkt. Durch ein Byte des sogenannten Video-RAMs wird nämlich die Vorder- und die Hintergrundfarbe für 64 Punkte bestimmt. Diese 64 Punkte ergeben sich aus einem untereinanderliegenden 8-Byte-Block. (Siehe Kapitel 2.4.1.1), in dem alle gesetzten und alle nicht gesetzten Punkte jeweils eine einheitliche Farbe besitzen. Als Farbspeicher wird in der HGR der normale Video-RAM benutzt und zwar in der Art, daß die untersten 4 Bits (0-3) für die

Hintergrundfarbe, also die nicht gesetzten Punkte und die oberen 4 Bits (4-7) für die Farbe der 1-Bits zuständig sind. Das normale Farb-RAM, sowie das Register 33 des VIC's, haben in der HGR keine Funktion. Den Zusammenhang zwischen Grafikspeicher und Video-RAM soll folgendes Diagramm noch einmal verdeutlichen. Hierbei wurde für den Grafikspeicher die Basisadresse 8192 ($4000) und für das Video-RAM die Basisadresse 1024 ($0400) angenommen:

Grafikspeicher:

	Spalte 0		Spalte 1	
	Reihe	Adresse	Reihe	Adresse
	0	8192	0	8200
	1	8193	1	8201
	2	8194	2	8202
Zeile 0	3	8195	3	8203
	4	8196	4	8204
	5	8197	5	8205
	6	8198	6	8206
	7	8199	7	8207
	0	10752	0	10760
	1	10753	1	10761
	2	10754	2	10762
Zeile 1	3	10755	3	10763
	4	10756	4	10764
	5	10757	5	10765
	6	10758	6	10766
	7	10759	7	10767

Video-RAM:

	Spalte 0		Spalte 1	
	Reihe	Adresse	Reihe	Adresse
	0	1024	0	1025
	1	1024	1	1025
	2	1024	2	1025
Zeile 0	3	1024	3	1025
	4	1024	4	1025
	5	1024	5	1025
	6	1024	6	1025
	7	1024	7	1025
	0	1064	0	1065
	1	1064	1	1065
	2	1064	2	1065
Zeile 1	3	1064	3	1065
	4	1064	4	1065
	5	1064	5	1065
	6	1064	6	1065
	7	1064	7	1065

Die beiden Grafiken stellen an einem konkreten Beispiel den Zusammenhang zwischen Grafikspeicher und Video-RAM dar. Die obere Tabelle stellt einen Ausschnitt des Grafikspeichers und die untere Tabelle einen Ausschnitt des dazugehörigen Video-RAMs dar. Besonders gut erkennt man, daß jeweils acht Bytes des Grafikspeichers einem Byte des Video-RAMs zugeordnet sind.

Die Adresse, in der die Farbe für einen Punkt festgelegt wird, läßt sich nach der folgenden Formel recht einfach berechnen. In der Formel steht 'X' für die X-Koordinate (0-319), 'Y' für die Y-Koordinate (0-199) und 'VR' für die Basisadresse des Video-RAMs (Standard 1024/$0400).

Die Spaltenadresse (0-39) erhalten wir einfach durch die 'glatte' Division der X-Koordinate (0-319) durch 8, da eine Reihe ja aus 40 Bytes zu je acht Bits besteht (8*40 = 320).

Spalte = INT (X/8)

Die Y-Koordinate errechnen wir nach demselben Schema. Wir müssen lediglich das Ergebnis noch mit 40 multiplizieren, da eine Zeile ja bekanntlich aus 40 8*8 Blöcken besteht.

Zeile = 40 * INT (Y/8)

In dem am Ende dieses Abschnitts folgenden Demoprogramm, finden Sie noch einmal die Formeln zur Punktberechnung und Farbsetzung in einzelnen Unterprogrammen abgelegt. Sie sollten diese Routinen getrennt abspeichern, damit sie auch später noch in eigenen Programmen verwendet werden können.

Nun zu unserem Programm: In dieser kleinen Demonstration wird auf später erläuterte Weise eine Ellipse auf dem Grafikbildschirm gezeichnet. Die Punkt- und Farbsetzung geschieht nach den gerade abgeleiteten Formeln. Lediglich die beiden Befehle in Zeile 140 sollten noch genauer besprochen werden. Eigentlich haben die Befehle 'DIM' und 'CLR' mehr mit der Datenverarbeitung als mit der Grafik zu tun. Allerdings nutzen wir in diesem Fall diese Befehlskombination, um unseren Grafikspeicher zu löschen, denn wenn wir einen Array mit 'DIM' dimensionieren, so wird diesem Array ein Speicherbereich zugewiesen und dieser gelöscht. Da nun unsere Grafik am BASIC-speicherende steht, wird diese mit 0-Bytes überschrieben, d. h. gelöscht. Diesen Trick sollten Sie sich merken, da eine vergleichbare Löschroutine in BASIC eine ganze Ecke langsamer ist.

Programmlisting:

```
100 rem **********************
110 rem *   hgr-plot demo    *
120 rem **********************
130 :
```

```
140 dim a$(10000) : clr :   rem bildschirm loeschen
150 :
160 v = 53248 : bs = 8192 : rem basisadressen
170 :
180 poke 53280,0 :          rem farbe
190 :
200 poke v+17,peek (v+17) or 32 : rem grafik an
210 poke v+24,24 :                rem grafik bei 8192
220 :
230 print chr$ (147); :           rem clearscreen
240 :
250 for i=0 to 6.28 step .01
260 :   x = 100*cos (i) + 160 : rem kreis x-koordinate
270 :   y =  50*sin (i) + 100 : rem kreis y-koordinate
280 :   fa=6+1*16 : rem farbe weiss auf blau
290 :   gosub 400 : rem punkt / farbe setzen
300 :   gosub 500 : rem farbe setzen
310 next i
320 :
330 wait 198,1 : poke 198,0 : rem auf taste warten
340 :
350 poke v+17,peek (v+17) and not 32 : rem textmodus an
360 poke v+24,21 :                     rem alter wert
370 :
380 end : rem programmende
390 :
400 rem ********************
410 rem *   punkt setzen   *
420 rem ********************
430 :
440 xa = x and 504 :                   rem x-adresse
450 ya = int (y/8) * 320 + (y and 7) : rem y-adresse
460 ad = bs + xa + ya :                rem adresse
470 ma = 2^(7-(x and 7)) :             rem or-maske
480 poke ad,peek (ad) or ma :          rem punkt setzen
490 :
500 rem ********************
510 rem *   farbe setzen   *
520 rem ********************
530 :
```

```
540 sp = int (x/8) :       rem spalte (0-39)
550 ze = int (y/8) * 40 : rem zeile (0-24)
560 ad = 1024 + sp +ze :  rem adresse
570 poke ad,fa :          rem farbe setzen
580 return :              rem zurueck
```

2.4.1.3 Die Farbgebung und Punktsetzung in der MC

Wahrscheinlich haben Sie sich auch schon darüber geärgert, daß Sie in der HGR in einem 8*8 Block nur zwei verschiedene Farben setzen können (einmal die für diesen Block definierte Punktfarbe und die blockspezifische Hintergrundfarbe). Doch es geht auch anders: Ist nämlich Bit 4 im VIC-Register 22 gesetzt, so können Sie in einem 8-Byte-Block maximal vier verschiedene Farben (inklusive der allgemeinen Hintergrundfarbe) verwenden. Trotz dieser erhöhten Farbauflösung belegt unser Grafikspeicher immer nur noch ca. 8 KByte im Speicher. Wie ist dies möglich ? Nun, die erhöhte Farbauflösung geht auf Kosten der Punktauflösung, denn diese beträgt jetzt nur noch 160*200 Punkte. Da das Bild jedoch trotz halber Punktzahl in X-Richtung keine Anstalten macht zu schrumpfen, wird jeder Punkt in X-Richtung doppelt so breit dargestellt. Dementsprechend wird jeder einzelne Punkt durch zwei Bits repräsentiert. Sind beide Bits gleich Null so wird der Punkt in der Hintergrundfarbe dargestellt. Die Farbnummer für diese Hintergrundfarbe stammt jetzt aus dem VIC-Register 33. Ist das zweite Bit gesetzt und das erste dagegen auf Null (0/1), so wird die Farbe aus den Bits 4-7 des Video-RAMs gebildet. Sind die beiden Bits genau verdreht (1/0), so wird die Farbe des Punktes aus den unteren 4 Bits (0-3) des Video-RAMs gebildet. Sind beide Bits gesetzt, so stammt die Farbe aus dem Farb-RAM. In einer Tabelle soll das Ganze nun veranschaulicht werden:

Modus 0 : 00 = Hintergrundfarbe 0 (Register 33)
Modus 1 : 01 = Video-RAM Bits 4-7
Modus 2 : 10 = Video-RAM Bits 0-3
Modus 3 : 11 = Farb-RAM

(Der Zeichenmodus ist eine Vereinbarung unsererseits, die zur besseren Bezeichnung der einzelnen Bitkombinationen dient.)
Logischerweise reicht die in Kapitel 2.4.1.1 beschriebene Formel zur Punktberechnung und Farbsetzung nicht mehr aus, so daß wir hier die erweiterte Form erarbeiten wollen.
Da sich an der Y-Auflösung gegenüber der HGR nichts geändert hat, bleibt die Formel zur Berechnung der relativen Adresse der horizontalen Reihe identisch, nämlich:

Y-Reihenadresse = INT (Y/8) * 320 + (Y AND 7)

Bei der Berechnung der X-Reihe dagegen ändert sich ein wenig. Sie lautet jetzt:

X = 2 * INT (X)

und: Bytenummer=INT (X/8)*8 oder Bytenummer=X AND 504

Wie Sie sehen, bringen wir die X-Koordinate wieder auf den Bereich 0-319, der jetzt allerdings in Schritten zu je zwei Punkten aufgeteilt ist, da ja immer zwei Bits einen Punkt darstellen. Schwierig wird es erst bei der Punktsetzung, da wir aufgrund des gewählten Modus möglicherweise die zwei Bits verschieden voneinander setzen müssen. Da ein Punkt jetzt durch zwei Bits dargestellt wird, ist es notwendig, eine Maske zu entwickeln. mit der es möglich ist, diese beiden Bits auf einmal zu beeinflussen. Daher gehen wir folgendermaßen bei der Berechnung der Maske vor. Zuerst errechnen wir den Wert des ersten Bits nach folgender, uns altbekannter, Formel:

Maske1 = 2^(7-(X AND 7))

Den Wert des zweiten Bits könnten wir jetzt auf die gleiche Art und Weise errechnen, doch es geht auch einfacher. Da der

Exponent der, durch das zweite Bit gebildeten Zweierpotenz um eins kleiner ist, als der des ersten Bits, können wir schreiben:

```
Maske1 = 2^(7-(X AND 7))
Maske2 = Maske1 + Maske1 / 2 oder Maske2 = 1.5 * Maske1
```

In einer Formel zusammengefasst, sieht das so aus:

```
Maske = 1.5*2^(7-(X AND 7))
```

Beachten Sie bitte, daß für X durch obige Rechnung nur gerade Werte vorkommen. Wollen wir nun einen Punkt mittels der 'POKE'-Anweisung setzen, so müssen wir erst die beiden betroffenen Bits löschen und dann die dem jeweiligen Modus entsprechende Maske mittels einer ODER-Verknüpfung hinzufügen. Die Maske die hinzugefügt werden muß, erhalten wir durch eine weitere Maskierung unseres Bitpaares mit der passenden Matrix. Doch nun zur Farbsetzung in der MC:
Die Farbsetzung in der Multicolorgrafik ist in etwa genauso komplex, wie auch schon die Punktsetzung in der MC. Denn auch bei der Farbsetzung läßt es sich nicht umgehen, mit Masken zu arbeiten, will man nicht Gefahr laufen, vor dem Monitor einzuschlafen. Genau wie in der HGR berechnen wir auch hier wieder Zeile (0-24) und Spalte (0-39) und berechnen hiermit die Adresse, in der die Farbe festgelegt werden muß. Um die PLOT-Routine nicht künstlich zu verlangsamen, führen wir die möglicherweise notwendige Addition von 54272 ($D400) um die Farbadresse zu erhalten in der 'POKE'-Anweisung durch, die jetzt so lautet:

```
SP = INT (X/8) : ZE = INT (Z/8) * 40
AD = 1024 + SP + ZE
POKE AD,(PEEK (AD) AND FA(M)) OR FA-(FA*15)*(M=1)
```

Damit Sie nicht völlig verzweifeln, folgt hier die Erklärung der Formel:

Als erstes löschen wir die für die Farbe zuständigen Bits, damit der Farbkode nicht durch zufällig noch gesetzte Bits verfälscht werden kann und fügen dann mittels einer ODER-Verknüpfung die Farbe hinzu. Einer genaueren Erklärung bedarf wahrscheinlich nur noch die Anweisung (M=1). Ist der Modus (M) = 1, so liefert die Anweisung das Ergebnis (-1), andernfalls 0. Ist der Modus (M) also = 1, so wird zu dem Farbkode (FA) noch (-15)*FA*(-1), also 15*FA, hinzuaddiert, wodurch wir dann FA*16, also die oberen vier Bits beeinflussen.

Demoprogramm:

```
100 rem **********************************
110 rem *   multicolor demoprogramm     *
120 rem **********************************
130 :
140 dim a$ (5000) : clr : rem grafikspeicher loeschen
150 :
160 ma (0)=  0 : rem %00000000 modus 0
170 ma (1)= 85 : rem %01010101 modus 1
180 ma (2)=170 : rem %10101010 modus 2
190 ma (3)=255 : rem %11111111 modus 3
200 :
210 fa (0)=  0 : rem %00000000 modus 0
220 fa (1)= 15 : rem %00001111 modus 1
230 fa (2)=240 : rem %11110000 modus 2
240 fa (3)=  0 : rem %00000000 modus 3
250 :
260 v=53248 : rem basisadresse vic
270 :
280 poke 53280,0 : poke 53281,0 : rem farben setzen
290 :
300 poke v+17,peek (v+17) or 32 : rem grafik an
310 poke v+24,24                : rem grafikspeicher 8192
320 poke v+22,peek (v+22) or 16 : rem mc-grafik
330 :
340 for j=0 to 30 step 3 :            rem radien
350 :   m=(m+1) and 3 : if m=0 then 350 : rem modus +1
360 :   read fa : rem farbe lesen
```

```
370 :    for i=.02 to 6.28 step .02 : rem ellypse zeichnen
380 :        x=(40-j)*cos (i)+80 :    rem x-koordinate 0-160
390 :        y=(40-j)*sin (i)+100 :   rem y-koordinate 0-199
400 :        gosub 540 : rem punkt mit farbe setzen
410 :    next i : rem schleifenende 'i'
420 next j :     rem schleifenende 'j'
430 :
440 wait 198,1 : poke 198,0 : rem auf taste warten
450 :
460 poke v+17,peek (v+17) and not 32 : rem grafik aus
470 poke v+24,21 :                     rem alter wert
480 poke v+22,peek (v+22) and not 16 : rem mc-modus aus
490 :
500 end : rem programmende
510 :
520 data 2,4,5,7,8,10,13,14,2,12 : rem farben
530 :
540 rem ****************************
550 rem *   punktberechnung (mc)    *
560 rem ****************************
570 :
580 ya=int (y/8)*320+(y and 7) : rem reihenanfang
590 x=2*int (x) :                rem x-bereich 0-320
600 xa=x and 504 :               rem xa=int (x/8)*8
610 ad=8192+ya+xa :              rem byteadresse
620 :
630 ma=1.5*2^(7-(x and 7)) :     rem maske
640 :
650 poke ad,peek (ad) and (255-ma) or (ma and ma(m))
660 :
670 rem *******************
680 rem *   farbsetzung   *
690 rem *******************
700 :
710 if m=0 then return : rem keine farbe
720 sp=int (x/8) : ze=int (y/8)*40 : rem spalte / zeile
730 ad=1024+sp+ze :                  rem adr. im videoram
740 :
```

```
750 poke ad,(peek (ad) and fa(m)) or fa-(fa*15)*(m=1)
760 :
770 return : rem zum hauptprogramm
```

2.4.2 Die Sprites

Außer dem Textmodus und der Einzelpunktgrafik, kennt der sowohl im Commodore 64 als auch im Commodore 128 in verbesserter Form eingebaute Videocontroller (VIC) noch die sogenannten Sprites. Sprites sind kleine eigenständige Grafiken, die sowohl vor, als auch hinter dem eigentlichen Bildschirminhalt liegen können. Sie sind frei beweglich und können durch spezielle Register an jede beliebige Stelle des Bildschirms gebracht werden. Von diesen Sprites besitzt der VIC II acht an der Zahl, die sich alle voneinander in Form, Farbe, Größe, Lage und Priorität unterscheiden können. Durch die Sprites eröffnen sich vor allem dem Programmierer von Spielen ungeahnte Möglichkeiten. Ist es doch mittels der Sprites ein Leichtes, Kollisionen festzustellen und Objekte ohne großen Aufwand zu bewegen. Doch nicht nur für Spiele sind diese kleinen Dinger hervorragend geeignet, so kann man sie beispielsweise auch in der Auswertung von Messreihen oder als Markierung in der kommerziellen Grafikerstellung benutzen.

Mit dem Debüt des Commodore 64 setzte die Firma Commodore Ltd. neue Maßstäbe auf dem Homecomputer-Sektor. Für diese Erhöhung des Homecomputerniveaus waren natürlich auch die Sprites und die gesamte HI-RES kein unwesentlicher Faktor. Doch nun zu der Programmierung der Sprites. Da die Sprites in Maschinensprache sowieso nur "zu Fuß" programmiert werden können und ihnen im BASIC 7.0 jede Menge leistungsfähige Befehle zur Spriteprogrammierung zur Verfügung stehen, laufen alle folgenden Programme im 64er Modus und benutzen ausschließlich den altbewährten POKE-Befehl. Doch auch wem dies zu umständlich erscheint, und wer sowieso nicht vorhat, die

Sprites in Assembler zu programmieren, sollte die nun folgenden Kapitel ruhig lesen, denn in ihnen werden die Grundlagen für die allgemeine Spriteprogrammierung auch in BASIC geklärt.

2.4.2.1 Form, Lage und Verwaltung von Sprites

Es ist allseits bekannt: Der VIC II kann acht verschiedene Sprites darstellen. Diese Sprites können alle eine andere Form besitzen, müssen also einen für sie speziell definierten Speicherraum besitzen. Jeder dieser Speicherräume ist 63 Bytes lang, belegt aber 64 Bytes im Speicher, da der 16K große VIC-Adreßraum in 256 Blöcke zu je 64 Bytes unterteilt wird (3*21+1=64). Um nun dem VIC mitzuteilen, aus welchem dieser Speicherblöcke er die Sprite-Matrix beziehen soll, existiert für jedes Sprite ein sogenannter Pointer. Diese acht Pointer (0-7) belegen immer die letzten acht Bytes des Video-RAMs, also in der Normalkonfiguration die Adressen 2040 bis 2047 (Sprites 0-7). Sollte das Video-RAM verschoben worden sein, so sind auch die Spritepointer verschoben. Die mit Hilfe der Pointer definierten Adreßräume beziehen sich immer relativ auf den Beginn des VIC-Adreßraums. Es muß also immer die momentane Startadresse des VIC-Adreßraums zu der durch den jeweiligen Pointer definierten Blockstartadresse hinzuaddiert werden, um die Adresse eines Sprite-Blocks zu erhalten. Auch für die Sprites gelten die in Kapitel 2.1 besprochenen Ausnahmen ($1000-$1FFF/$9000-$9FFF). Doch nun zum Aufbau eines Sprites:

Ein Sprite besteht aus 21 Reihen mit je 24 Punkten. Dies hat zur Folge, daß die Matrix nicht wie in der HI-RES abgelegt sein kann, da 21 nicht ohne Rest durch 8 zu dividieren ist. Vielmehr liegt jetzt ein Byte neben dem anderen und repräsentiert durch seine Bits wieder jeweils acht Punkte. Man könnte ein Sprite also folgendermaßen in drei Spalten aufteilen:

```
              Spalte 1   Spalte 2   Spalte 3
------------------------------------------------------
Zeile  1 : 76543210 | 76543210 | 76543210 | Bytes  1 -  3
Zeile  2 : 76543210 | 76543210 | 76543210 | Bytes  4 -  6
Zeile  3 : 76543210 | 76543210 | 76543210 | Bytes  7 -  9
Zeile  4 : 76543210 | 76543210 | 76543210 | Bytes 10 - 12
Zeile  5 : 76543210 | 76543210 | 76543210 | Bytes 13 - 15
Zeile  6 : 76543210 | 76543210 | 76543210 | Bytes 16 - 18
Zeile  7 : 76543210 | 76543210 | 76543210 | Bytes 19 - 21
Zeile  8 : 76543210 | 76543210 | 76543210 | Bytes 22 - 24
Zeile  9 : 76543210 | 76543210 | 76543210 | Bytes 25 - 27
Zeile 10 : 76543210 | 76543210 | 76543210 | Bytes 28 - 30
Zeile 11 : 76543210 | 76543210 | 76543210 | Bytes 31 - 33
Zeile 12 : 76543210 | 76543210 | 76543210 | Bytes 34 - 36
Zeile 13 : 76543210 | 76543210 | 76543210 | Bytes 37 - 39
Zeile 14 : 76543210 | 76543210 | 76543210 | Bytes 40 - 42
Zeile 15 : 76543210 | 76543210 | 76543210 | Bytes 43 - 45
Zeile 16 : 76543210 | 76543210 | 76543210 | Bytes 46 - 48
Zeile 17 : 76543210 | 76543210 | 76543210 | Bytes 49 - 51
Zeile 18 : 76543210 | 76543210 | 76543210 | Bytes 52 - 54
Zeile 19 : 76543210 | 76543210 | 76543210 | Bytes 55 - 57
Zeile 20 : 76543210 | 76543210 | 76543210 | Bytes 58 - 60
Zeile 21 : 76543210 | 76543210 | 76543210 | Bytes 61 - 63
```

Wie definieren wir nun ein Sprite, z. B. ein Raumschiff? Nun, die erste Möglichkeit, die sich uns bieten würde, wäre die Benutzung des eingebauten Spriteeditors. Eine andere Möglichkeit bietet sich uns, indem wir die Sprites von Hand auf einem Blatt Papier definieren. Da diese Buch speziell die Grundlagen klären soll und keine BASIC-Anleitung ist, gehen wir nur auf die zweite Form genauer ein. Suchen Sie einen komfortableren Sprite-Editor, als den eingebauten, so empfehle ich Ihnen, einmal das Angebot für den C-64 zu testen. Ich bin sicher, daß dieser auch Ihren Anspruchen genügt. Doch nun wieder zum Thema: Wie definieren wir also ein Sprite von Hand? Eigentlich ist dies recht einfach: Wir nehmen also ein Blatt Papier, zeichnen unsere drei Spalten mit jeweils acht Kästchen ein und füllen diese nach unseren Wünschen. Dann rechnen wir die 63 Bytes folgendermaßen aus: Da ein ausgemaltes Kästchen einem

gesetzten Bit entspricht, müssen wir einfach nichts weiter tun, als die den gesetzten Bits zugeordneten Werte zu addieren, und schon haben wir unser Byte. Wenn wir nun so bei allen 63 Bytes vorgehen, erhalten wir unser Sprite in Form der sogenannten DATA's. Zum besseren Verständniss das Ganze noch einmal bildlich:

```
|7|6|5|4|3|2|1|0|
-----------------
|1|0|1|0|1|0|1|0| = 2^7+2^5+2^3+2^1
                  = 128+32+8+2
                  = 170
```

Wie das oben Gesagte z. B. aussehen kann, zeigt Ihnen das folgende Beispielprogramm, das Sie ruhig abtippen sollten, sofern Sie nicht die Diskette zum Buch besitzen. Es lohnt sich bestimmt für Sie, denn Sie lernen praktisch die Handhabung der einzelnen Spritesteuerungsregister.

```
100 rem ******************************
110 rem *   sprite-demonstration    *
120 rem ******************************
130 :
140 print chr$ (147); : rem clear screen
150 :
160 v=53248 : rem basisadresse vic
170 bl=11 :    rem blocknummer
180 x=300 :    rem x-koordinate
190 y=100 :    rem y-koordinate
200 :
210 poke 2040+4,bl : rem block fuer sprite 4
220 :
230 for i=bl*64 to bl*64+62
240 :   read a :   rem dataelement lesen
250 :   poke i,a : rem im block ablegen
260 next i
270 :
```

```
770 data   0,  0,  0 : rem ........ ........ ........
780 data   0,  0,  0 : rem ........ ........ ........
790 data   0,  0,  0 : rem ........ ........ ........
800 data   0,  0,  0 : rem ........ ........ ........
810 data   0,  0,  0 : rem ........ ........ ........
820 data   0,  0,  0 : rem ........ ........ ........
830 data   0,  0,  0 : rem ........ ........ ........
840 data   0,  0,  0 : rem ........ ........ ........
850 data   0,  0,  0 : rem ........ ........ ........
860 data   0,  0,  0 : rem ........ ........ ........
870 data   0,  0,  2 : rem ........ ........ ......*.
880 data   0,  0, 10 : rem ........ ........ ....*.*.
890 data   0,  0, 42 : rem ........ ........ ..*.*.*.
900 data   0,192,170 : rem ........ **...... *.*.*.*.
910 data   3, 50,170 : rem ......** ..**..*. *.*.*.*.
920 data  10,170,160 : rem ....*.*. *.*.*.*. *.*.....
930 data  42,170,165 : rem ..*.*.*. *.*.*.*. *.*..*.*
940 data 170,170,175 : rem *.*.*.*. *.*.*.*. *.*.****
950 data 171,255,229 : rem *.*.*.** ******** ***..*.*
960 data  42,170,160 : rem ..*.*.*. *.*.*.*. *.*.....
970 data  10,170,170 : rem ....*.*. *.*.*.*. *.*.*.*.
```

Geben Sie bitte dieses Programm ein und speichern Sie es auf Diskette ab. Wir werden sämtliche Spriteeigenschaften anhand dieses Programms erarbeiten.

2.4.2.2 Positionierung von Sprites

Jedes Sprite kann über 512 X-Positionen und über 256 Y-Positionen bewegt werden. Dabei geben die Koordinaten immer die Position der linken, oberen Ecke des jeweiligen Sprites an. Soll das Sprite in der linken, oberen Ecke des Bildschirms liegen, so muß es bereits die Koordinaten X=24 und Y=50 besitzen. Ein Sprite kann damit auch außerhalb des eigentlichen Bildschirms positioniert werden. Welche Bedeutung dies für die Programmierung hat, werden wir weiter unten noch sehen. Es ergeben

sich also folgende Umrechnungen der Spritekoordinaten in die Bildschirmkoordinaten (X:0-40/Y:0-24):

Spalte = (X+24)/40 ; Zeile = (Y+50)/25

Wie Sie vielleicht schon der Registerbeschreibung entnommen haben, existieren für die X- und für die Y-Koordinaten jedes Sprites jeweils zwei Register für jedes Sprites mit denen das Sprite über einen Raum von 256 X- und 256 Y-Koordinaten bewegt werden kann. Doch wie sprechen wir die weiteren 256 X-Koordinaten an? Für diesen Zweck existiert das VIC-Register 16. Es stellt für jedes Sprite das 8. Bit der X-Koordinate dar (2^8 = 256). Ist das korrespondierende Bit gesetzt, so addiert der VIC also den Wert 256 zur eigentlichen X-Koordinate.
Beispiel : Wir wollen das Sprite mit der Nummer vier an die Koordinaten x=300 und y=100 setzen. Dazu gehen wir folgendermaßen vor, um das Sprite an die besagte Position zu setzen:

1. Wir schreiben in das Register 8 (X-Koordinate von Sprite 4) den 8-Bit Anteil der X-Koordinate.

Also: POKE V+8,X AND 255

2. Wir setzen, wenn es notwendig ist, das 8. Bit der X-Koordinate in Register 16 (MSB's der X-Koordinaten). Und das geht so :

POKE V+16,PEEK (V+16) OR (SGN(X AND 256) * 2^4)

Wir lesen also zuerst den momentanen Wert aus und fügen dann unseren neuen Wert mittels einer ODER-Verknüpfung hinzu, um die anderen Bits nicht zu beeinflussen. Den Wert der hinzugefügt werden muß, errechnen wir nun folgendermaßen : Ist die X-Koordinate größer als 255, so ergibt die SGN-Funktion den Wert 1, und es wird der Wert 1*2^4 (also 16) hinzugefügt. Ist der X-Wert allerdings kleiner als 256, so ergibt die SGN Funktion den Wert 0, und wir belassen Register 16 in seiner alten Form. Was aber würde passieren, wenn das Bit schon auf 1

stand? Es würde ganz einfach nicht gelöscht werden! Also müssen wir die Formel noch etwas erweitern. Sie lautet jetzt:

POKE V+16,(PEEK (V+16) AND NOT 2^4) OR (SGN (X AND 256) * 2^4)

(Diese Formel kann auch dann angewendet werden, wenn die X-Koordinate kleiner als 256 ist.)

3. Wir setzen die Y-Koordinate im VIC-Register 9 wie folgt:

POKE V+9,Y

Falls Sie dies jetzt eingegeben haben sollten, sehen Sie natürlich noch nichts, da das Sprite ja noch nicht aktiviert war. Aber erweitern wir unser Grundprogramm trotzdem um die folgenden Programmzeilen und speichern es wieder ab :

```
280 poke v+8,x and 255 : rem x-koordinate
290 poke v+16,(peek (v+16) and not 2^4) or (sgn (x and 256) * 2^4)
300 poke v+9,y and 255 : rem y-koordinate
310 :
```

2.4.2.3 Aktivieren der Sprites

Zum Ein- bzw. Ausschalten der Sprites stellt uns der VIC II das Register 21 zur Verfügung. Ist das mit dem jeweiligen Sprite korrespondierende Bit gesetzt, so ist auch das Sprite aktiviert. Übersichtlich dargestellt sieht die Beziehung der einzelnen Bits zu den dazugehörigen Sprites so aus:

Bit	: 7	6	5	4	3	2	1	0
Sprite	: 7	6	5	4	3	2	1	0
Wert	: 128	64	32	16	8	4	2	1

Diesen Aufbau finden wir bei fast allen auf den folgenden Seiten beschriebenen Registern, die mit der Veränderung der Spriteeigenschaften durch den VIC II zu tun haben, so z. B. bei den Registern 23 und 29 zur Spritevergrößerung oder auch bei dem Register 16. Bedingt durch diese Kodierung können wir nicht einfach schreiben:

POKE V+21,2^4 ; (V ist die Basisadresse des VIC!)

um Sprite vier einzuschalten, da hierdurch alle anderen Sprites wieder ausgeschaltet würden. Wir müssen also dafür sorgen, daß der Zustand der übrigen Bits nicht verändert wird. Um nun Sprite vier einzuschalten, sollte man also besser folgendermaßen vorgehen:

POKE V+21,PEEK (V+21) OR 2^4

Analoges gibt man ebenfalls beim Ausschalten der einzelnen Sprites ein. Man benutzt dann allerdings die "AND NOT" Kombination, die das angegebene Bit löscht, alle anderen Bits aber im alten Zustand beläßt. Um also Sprite vier auszuschalten, ist es vorteilhaft zu schreiben:

POKE V+21,PEEK (V+21) AND NOT 2^4

Will man mehrere Sprites gleichzeitig ausschalten, schreibt man entweder:

POKE V+21,PEEK (V+21) AND (255-2^4-2^5)

oder:

POKE V+21,PEEK (V+21) AND NOT (2^4 OR 2^5)

Hierbei wäre die zweite Möglichkeit der ersten vorzuziehen, da bei ihr auch ruhig zwei gleiche Werte innerhalb der Klammern angegeben werden dürfen. Fügen wir also jetzt folgende Programmzeile in unser Programm ein und starten es. Falls die Spritefarbe nicht mit der Hintergrundfarbe identisch ist, müßten Sie jetzt ein einem Raumschiff mehr oder minder gleichendes Gebilde erkennen können. Hier nun die einzufügende Programmzeile um Sprite vier zu aktivieren:

```
320 poke v+21,peek (v+21) or 2^4 : rem sprite an
```

2.4.2.4 Farbgebung bei Sprites

Wie Sie wissen sollten, kann jedes Sprite eine von 16 verschiedenen Farben besitzen. Hierfür existiert für jedes Sprite ein eigenes Register im Videocontroller. Es sind dies die Register 39 bis 46 für die Sprites null bis sieben.
Erinnern Sie sich noch an den Multi-Color-Modus in der HIRES? Auch bei den Sprites brauchen Sie hierauf nicht zu verzichten, denn er ist auch hier, allerdings in etwas abgewandelter Form, vorhanden.
Soll ein Sprite im Multi-Color-Modus dargestellt werden, so müssen nicht alle Sprites im Multi-Color-Modus dargestellt werden, sondern es kann im Register 28 festgelegt werden, in welchem Modus das Sprite dargestellt werden soll. Der Aufbau des Registers 28 entspricht im übrigen dem des weiter oben beschriebenen Registers 21. Ist also ein Bit in Register 28 gesetzt, so wird das diesem Bit zugeordnete Sprite im Multi-Color-Modus dargestellt. Wie auch in der Multi-Color-Grafik (MC) halbiert sich auch bei den Sprites die Punktauflösung zugunsten der jetzt verdoppelten Farbauflösung. Ein Punkt wird

jetzt genauso wie bei der Multi-Color-Grafik durch zwei Bits folgendermaßen kodiert:

00 = Hintergrundfarbe 0 (Register 33)
01 = Sprite-Multicolorfarbe 0 (Register 37)
10 = Sprite-Multicolorfarbe 1 (Register 38)
11 = Normale Spritefarbe (Register 39-46)

Sorgen wir also dafür, daß unser Sprite im Multicolormodus dargestellt wird. Als Spritenormalfarbe definieren wir hier blau, für die beiden Multicolorfarben setzen wir gelb und rot. Hier nun der einzufügende Programmteil:

```
330 poke v+28,peek (v+28) or 2^4 : rem multicolor
340 :
350 poke v+43,6 : rem normalfarbe
360 poke v+37,2 : rem multicolorfarbe 1
370 poke v+38,7 : rem multicolorfarbe 2
380 :
```

2.4.2.5 Sprite-Prioritäten

Gehen wir davon aus, daß sich auf unserem Bildschirm mehrere Sprites, ein bißchen Text und ein paar Grafikzeichen befinden. Bewegen wir nun die Sprites, so ergeben sich für den VIC erst dann Probleme, wenn ein Sprite dieselbe Position, wie ein anderer Bildschirminhalt annimmt. Aus diesem Grund existiert ein Register im Videocontroller, in dem er bei einer Überlappung nachschauen kann, ob das Sprite vor oder hinter dem anderen Bildschirminhalt liegen soll. Dieses besagte Register ist das VIC-Register 27. Ist das einem Sprite zugeordnete Bit in diesem Register gesetzt, so liegt es hinter dem anderen Bildschirminhalt; ist das Bit gelöscht, vor dem eigentlichen Bilschirminhalt. So weit nun schön und gut! Aber was passiert, wenn sich mehrere Sprites überlappen, da hierfür doch kein Register besteht? Nun, genau wie in der Schule eine Eins immer besser ist als eine

Zwei, kann man auch sagen, daß das Sprite, welches "besser" ist vor dem "schlechteren" liegt. Nun kennt der Computer natürlich nicht schlecht und gut, deshalb ist die Spriteinterne Priorität so definiert, daß immer das Sprite mit der niedrigeren Nummer vor dem mit der höheren Nummer liegt. Beispiel: Überlappen sich Sprite drei und fünf, so besitzen die gesetzten Punkte von Sprite drei die höhere Priorität. An den Stellen, an denen in Sprite drei kein Punkt gesetzt ist, besitzt dann entweder das Sprite fünf oder der eigentliche Bildschirminhalt die höhere Priorität, je nachdem, ob das Bit im Register 27 gesetzt oder gelöscht ist.

Wir fanden diese Eigenschaften so interessant, daß wir es uns nicht verkneifen konnten, noch ein kleines Demoprogramm zu schreiben, das natürlich nicht an das andere Demoprogramm angehängt werden darf, da dieses ja noch nicht vollständig ist. Zuerst einmal die Programmzeilen, die Sie in das alte Demoprogramm einfügen müssen:

```
390 poke53280,0 : poke53281,0 : poke646,5 : rem bildschirm
400 :
410 poke v+27,peek (v+27) and not 2^4 : rem vor hintergrund
```

Hier nun das spezielle Demoprogramm, das entgegen dem alten Demoprogramm im 128er Modus läuft. Wir wollen hier keine weiteren Erklärungen zum Programm geben, schauen Sie es sich doch einmal genau an, Sie werden dann schon die Funktionsweise erkennen. Nun dann, das spezielle Demoprogramm:

```
100 rem ******************************
110 rem *   sprite-prioritaet demo   *
120 rem ******************************
130 :
140 scnclr : rem bildschirm loeschen
150 :
160 fa=3.1415/180 : rem umrechnungsfaktor
170 :
180 for i=3648 to 3710 : rem block 57 fuer sprite 2
190 :   read a :   rem matrix lesen
200 :   poke i,a : rem und in puffer schreiben
```

```
210 next i
220 :
230 graphic 1,1 : rem grafik an und loeschen
240 :
250 circle 1,12,10,11,10 : rem kreis zeichnen (erde)
260 paint 1,12,10 :        rem und fuellen
270 sshape a$,0,0,23,20 :  rem in a$ uebernehmen
280 sprsav a$,3         :  rem und als sprite ablegen
290 :
300 circle 1,35,10, 6, 6 : rem kreis zeichnen (mond)
310 paint 1,35,10        : rem und fuellen
320 sshape a$,24,0,44,20 : rem in a$ speichern
330 sprsav a$,4 : sprsav a$,1 : rem und ablegen
340 :
350 sprite 3,1,7,0,1,1 : rem parameter fuer sprite drei
360 movspr 3,160,129 :   rem sprite drei setzen
370 sprite 2,1,8,0,1,1 : rem parameter fuer sprite zwei
380 movspr 2,160,129 :   rem sprite zwei setzen
390 :
400 color 0,1 : color 4,1 : color 1,8 : rem farben setzen
410 :
420 scnclr : rem hgr-screen loeschen
430 :
440 xr=120 : yr= 10 : rem ellipsen-radien
450 :
460 circle 1,158, 99,xr,yr : rem ellipse (umlaufbahn)
470 :
480 m=1 : xl=319 : rem modus, letzter x-wert
490 :
500 for w=0 to 360 step 2 : rem winkel setzen
510 :   x=xr*cos (w*fa)+170 : rem sprite-koordinate x
520 :   y=yr*sin (w*fa)+139 : rem sprite-koordinate y
530 :
540 :   movspr 1,x,y : movspr 4,x,y : rem sprites 1 & 4
550 :
560 :   if x<xl then m=1 : else : m=4 : rem flag umdrehen
570 :
580 :   sprite 4,sgn (m and 4),2 : rem sprite 4 an/aus
590 :   sprite 1,sgn (m and 1),2 : rem sprite 1 an/aus
600
```

```
610 :   xl=x : rem x-wert speichern
620 next w
630 :
640 goto 500 : rem programmende
650 :
660 rem ***************************
670 rem *   data's kontinente     *
680 rem ***************************
690 :
700 data  0,112,  0,  3,224,  0
710 data  7,224, 16, 15,192, 24
720 data 31,192, 60, 63,192,126
730 data 63,128,206, 31,  0, 31
740 data 28,128,255,  8,  1,255
750 data  4,  1,255,  3,  0,255
760 data  7,128, 63,  7,192, 62
770 data  7,128, 60,127,128,255
780 data  3,255, 16,  1,128,  0
790 data  0,192,  0,  0,  0,  0
800 data  0,  0,  0
```

2.4.2.6 Vergrößerung von Sprites

Wie Sie wissen, besteht ein Sprite aus 24*21 Punkten. Meist reicht diese Größe der Sprites aber nicht aus. Man hat dann die Auswahl zwischen der Zusammensetzung des Objektes aus mehreren Sprites oder der Vergrößerung des Sprites. Hierbei stellt die Vergrößerung meist die gewählte Lösung dar, da sich an der Sprite-Programmierung nichts Wesentliches ändert.
Außer der Formgestaltung durch die Punktmatrix eines Sprites, kann - wie schon gesagt - ein Sprite auch noch vergrößert werden. Dies ist sowohl in X-Richtung (Register 29) als auch in Y-Richtung (Register 23) getrennt möglich, wobei ein Punkt dann jeweils doppelt breit bzw. doppelt hoch dargestellt wird, das gesamte Sprite also entweder doppelt so breit, doppelt so hoch, oder beides zusammen wird. Soll ein Sprite in horizontaler Richtung vergrößert werden, so wird die Vergrößerung nach

rechts vorgenommen, wärend die Vergrößerung in vertikaler Richtung nach unten vorgenommen wird.Dies hat den Vorteil, daß die Koordinaten der linken, oberen Ecke sich bei einer Vergrößerung nicht verändern, die Routinen zur Spritepositionierung also nicht verändert werden müssen. Der Aufbau der beiden Register zur Spriteexpandierung entspricht übrigens dem Standardaufbau, trotzdem wollen wir ihn hier aber noch einmal darstellen:

```
Bit                 : 7  6  5  4  3  2  1  0
Vergrößertes Sprite : 7  6  5  4  3  2  1  0
```

Damit unser Demo-Raumschiff in X- und in Y-Richtung expandiert wird, fügen Sie bitte folgende Programmzeilen ein und speichern das Programm ab:

```
420 poke v+29,peek (v+29) or 2^4 :      rem expand x
430 poke v+23,peek (v+23) or 2^4 :      rem expand y
440 :
450 wait 198,1 : poke 198,0 : rem auf taste warten
460 :
```

2.4.2.7 Kollisionen

Vornehmlich in Spielen, seltener in kommerziellen Programmen, besteht das Bedürfnis, Kollisionen verschiedener Sprites untereinander oder mit anderen Bildschirminhalten zu kontrollieren. Für diesen Zweck stellt der VIC zwei Register zur Verfügung, mit denen es fast zum Kinderspiel wird, Actionspiele der verschiedensten Kategorien zu programmieren. Doch nun genauer zur Materie:

Kollisionen zwischen mehreren Sprites oder zwischen Sprites und Bildschirmzeichen lassen sich auf komfortable Weise mit dem Videocontroller ermitteln. Kollidieren beispielsweise die Sprites vier und sieben miteinander, so werden im Register 30 das

vierte (2^4=16) und das siebte (2^7=128) Bit gesetzt. Nach einer Kollision muß das Register allerdings wieder vom Programm zurückgesetzt werden, was aber schon durch das Auslesen des Registers möglich ist. Geschieht dies nicht, so kann dies dazu führen, daß weitere Kollisionen nicht mehr erkannt werden.

Wollen Sie hingegen feststellen ob ein Sprite mit einem Zeichen oder einem sonstigen Bildschirminhalt kollidiert ist, ob also beide mindestens in einem Punkt überlappen, so können Sie hierfür das Register 31 verwenden. Allerdings wird in diesem Register nur die Nummer desjenigen Sprites festgehalten, das kollidiert ist, nicht aber der Bildschirm- oder ASCII-Kode des Zeichens. Benötigen Sie diesen, so müssen Sie die Spritekoordinate, die ja auch die Kollisionskoordinate darstellt, so umrechnen, daß Sie einen Zeiger auf das im Video-RAM stehende Zeichen erhalten. Das Register 31 muß nach einer Kollision auf die oben beschriebene Art und Weise wieder zurückgesetzt werden. Die folgende Ergänzung des Demoprogrammes soll Ihnen noch einmal die Handhabung der Kollisionsregister verdeutlichen. In diesem Programmteil wird das Sprite von rechts nach links auf eine Wand zubewegt. Kollidiert es mit dieser, so fängt es an zu flimmern und wird schließlich desaktiviert. In den meisten käuflichen Spielen wird dieses Flimmern durch eine imposante Explosion ersetzt, deren Programmierung aber den Rahmen dieses Buches sprengen würde. Es wären einfach zu viele Spritedefinitionen nötig, um einen fliesenden Bewegungsablauf zu erreichen.

Fügen Sie also bitte jetzt die untenstehenden Programmzeilen ein. Hierdurch ist unser Demonstrationsprogramm dann komplett und schneidet alle Möglichkeiten der Spriteprogrammierung an. Am Ende dieses Kapitels über die Sprites wollen wir Sie noch einmal auffordern, doch alles auch einmal selber auszuprobieren. Der Lerneffekt ist wirklich gewaltig. Schreiben Sie doch einmal ein schönes Spiel, an Ideen wird es Ihnen sicherlich mangeln. Nur so ein Vorschlag: Schreiben Sie doch ein Spiel, bei dem Sie durch einen Tunnel fliegen. Die Routinen hierzu finden Sie bestimmt alle in diesem Buch oder auf der Diskette zum Buch, die hier nur wieder wärmstens empfohlen werden kann.

```
470 a=peek (v+31) : rem reg. 31 loeschen
480 print chr$ (19) : rem cursor home
490 :
500 for i=2 to 23 : rem balken zeichnen
510 :   print chr$ (18);" ";chr$ (146)
520 next
530 :
540 x=x-1 : rem sprite um 1 nach links
550 :
560 poke v+8,x and 255 : rem x-koordinate
570 poke v+16,(peek (v+16) and not 2^4) or (sgn (x and 256) * 2^4) : rem
msb
580 poke v+9,y : rem y-koordinate
590 :
600 if peek (v+31)=0 then 540 : rem kollision ?
610 :
620 for i=0 to 255
630 :   poke 2040+4,i : rem flimmern
640 next
650 :
660 poke 2040+4,bl :    rem alter block
670 :
680 for i=bl*64 to bl*64+62 step 3
690 :   poke i,0:poke i+1,0:poke i+2,0 : rem loeschen
700 :   for pa=1 to 50 : next pa : rem pause
710 next i
720 :
730 poke v+21,peek (v+21) and not 2^4 : rem sprite aus
740 :
750 end
760 :
```

2.4.3 Der Textmodus

Außer der HI-RES ist der VIC II noch in der Lage, eine 40*25 Zeichengrafik darzustellen. Dies ist der Ihnen sicherlich bekannte Textmodus.

Durch seine vielen Manipulationsmöglichkeiten wird der Textmodus häufiger angewendet als die HI-RES, da er im Gegensatz zum Grafikspeicher nur 1Kbyte im Speicher belegt. Aufgrund dieses acht mal kleineren Speichers, erhöht sich die Bearbeitungsgeschwindigkeit gegenüber der HI-RES um ein Vielfaches. Der Textmodus bietet genau wie die HI-RES den Normalmodus, in dem ein gesetztes Bit im Speicher auch als ein Punkt auf dem Bildschirm erscheint. Zusätzlich dazu gibt es noch den sogenannten Extended Color Modus, der in der HI-RES nicht existiert. Wie auch in der Grafik ist auch im Textmodus die Darstellung von Zeichen im Multi Color Modus möglich. Durch die Veränderung der Zeichensätze können auch mittels der Textgrafik hochauflösende Bilder erstellt werden. Dies wird vor allem bei Programmen genutzt, in denen es auf eine gute, farbige Grafik und eine möglichst hohe Bearbeitungsgeschwindigkeit ankommt. In erster Linie wird diese Betriebsart also in Spielen benutzt. Aber auch kommerzielle Dienstprogramme können einen Anwendungsbereich für Multicolorzeichen darstellen. Denken Sie hierüber doch einmal nach, Ihnen fallen bestimmt zwei drei Programmarten ein, bei denen Sie dieses Feature des Videocontrollers zu nutzen wüßten.

Wie Sie alle diese Features des Textmodus programmieren können, lesen Sie in den anschließenden Kapiteln.

2.4.3.1 Die Zeichensätze

Im Commodore 128 sind folgende acht Zeichensätze eingebaut:

Normal:	Groß/Grafik	Groß/Grafik revers
	Klein/Groß	Klein/Groß revers
Deutsch:	Groß/Grafik	Groß/Grafik revers
	Klein/Groß	Klein/Groß revers

Wie Sie sehen, sind die oben dargestellten acht Zeichensätze in zwei Hauptzeichensätze aufgeteilt: Dem normalen (bei gelöster CAPS-LOCK Taste) und dem deutschen Zeichensatz (bei eingerasteter CAPS-LOCK Taste). Zwischen diesen beiden Hauptzeichensätzen bestehen zwei Möglichkeit zur Umschaltung:

1. Die Umschaltung mittels der CAPS-LOCK Taste.

2. Die softwaremäßige Umschaltung (weiter unten beschrieben!).

Die im Kapitel 4- 3 des Handbuchs beschriebene softwaremäßige Variante funktioniert nicht, auf die dort beschriebene Weise! Mit dem Bit 6 des Prozessorports (Register 1 Zeropage), ist es lediglich möglich festzustellen, ob die Taste frei (0) oder arretiert (1) ist, vorausgesetzt, man ändert nicht Bit 6 im Register 0 auf 1, also auf Ausgabe. Wird dies allerdings gemacht, so funktioniert die Umschaltung doch, und die CAPS-LOCK Taste besitzt keine Funktion mehr. Die softwaremäßige Umschaltung funktioniert auch im 64er Modus!

Beim Drücken der CAPS-LOCK Taste kam es bei uns verschiedene Male zu einem Zeichenwirrwar auf dem Textschirm. Inwieweit dies auch für den anderen RAM zutrifft, konnten wir nicht überprüfen. Damit also alle Programme im 64er Modus 100% hard- und auch softwarekompatibel sind, muß die CAPS-LOCK Taste gelöst sein.

Zur Umschaltung zwischen den vier Unterzeichensätzen dient wieder das VIC-Register 24, das auch schon für die Lage des Grafikspeichers zuständig war. Nach dem Einschalten des Rechners befinden Sie sich immer im Groß/Grafik- und Groß/Grafik-revers Zeichensatz. Zwischen diesen beiden brauchen Sie nicht softwaremäßig umzuschalten, da nur beide auf einmal aktiviert sein können. Dies liegt daran, daß beide zusammen nur 256 Zeichen, also den Umfang des aktivierten Zeichensatzes, umfassen. Den Klein/Groß- bzw. den Klein/Groß-reversen Zeichensatz erreichen Sie durch eine Änderung im Register 24, was z. B. durch die Tastenkombina-

tion Commodore-Shift erfolgen kann. Eine nähere Beschreibung hierzu befindet sich im Kapitel 2.4.4.2.

2.4.3.2 Verschieben des Zeichensatzes

Die Verschiebung des Zeichensatzes läßt sich auf ähnliche Weise bewerkstelligen, wie auch die Verschiebung des Grafikspeichers. Allerdings kann hier die Verschiebung in wesentlich kleineren Schritten vorgenommen werden. Der Zeichensatz kann jetzt in minimal 2K Schritten (256 Zeichen) vorgenommen werden. Für die Verschiebung des Zeichengenerators sind nur die Bits 1-3 des Registers 24 relevant. Sie stellen die Adreßbits 11-13 (2048, 4096, 8192) für die Zeichenbasis dar, die sich genauso wie auch die Basisadresse des Grafikspeichers aus der Startadresse des VIC-Adreßraums plus der durch die Bits 1-3 kodierten Adresse ergibt (Siehe auch #2.2). Ist also Bit 1 im Register 24 gesetzt und der unterste 16K Block ausgewählt, so lautet die Adresse: 000000 + 2^11 = 2048 ($0800).

Theoretisch sind folgende Zeichensatzlagen möglich:

```
Reg.24 Bit 3  2  1  (Zeichensatzlage)
_____________________________________________________
           0  0  0  =    0 +    0 +    0 =     0 + BA
           0  0  1  =    0 +    0 + 2048 =  2048 + BA
           0  1  0  =    0 + 4096 +    0 =  4096 + BA
           0  1  1  =    0 + 4096 + 2048 =  6144 + BA
           1  0  0  = 8192 +    0 +    0 =  8192 + BA
           1  0  1  = 8192 +    0 + 2048 = 10240 + BA
           1  1  0  = 8192 + 4096 +    0 = 12288 + BA
           1  1  1  = 8192 + 4096 + 2048 = 14336 + BA
```

Die möglichen Basisadressen des VIC-Adreßraums (BA) entnehmen Sie bitte dem Kapitel #2.2, in dem die Grundlagen für die Verschiebung des Zeichengenerators, des Grafikspeichers

und des Video-RAMs geklärt werden. Wollen Sie also den Grafikspeicher nach 8192 ($2000) legen, so müssen Sie als erstes die beiden Bits 0/1 in der Adresse 56576 ($DD00) auf 1 setzen, um den ersten 16K- Block auszuwählen. Dann müssen Sie Bit drei im Register 24 setzen und die Bits eins und zwei löschen. Hierdurch wird bewirkt, daß nur Bit drei gesetzt bleibt, die Adresse des Zeichensatzes lautet dann also: 000000 + 2^13 = 8192 ($2000). In einem Programm könnte das so aussehen:

```
10 v=53248 : rem basisadresse vic
20 poke 56576,peek (56576) or 3 : rem block 0
30 poke v+24,(peek (v+24) and 241) or 2^3 : rem nur bit 3
```

Nachdem Sie dieses Programm gestartet haben, sehen Sie natürlich noch nichts, da ab der Adresse 8192 ($2000) ja kein Zeichensatz liegt. Kopieren wir also den Zeichensatz an die Adresse 8192. Zu diesem Zweck nun zuerst das Assemblerlisting mit BASIC-Lader, und im Anschluß daran stellen wir Ihnen dann die einfachere, aber langsamere BASIC-Äquivalente vor. Um noch einmal auf die BASIC-Lader zurückzukommen, prinzipiell stellen diese nichts anderes dar als auch das Maschinenprogramm, nur das mit ihm der Maschinenkode erst noch in den Speicher kopiert werden muß. Sie müssen also das Maschinenprogramm noch mit SYS aufrufen, falls der BASIC-Lader dies nicht für Sie erledigt. Da man des öfteren in die Verlegenheit kommt, Maschinenprogramme, neue Zeichensätze oder Sprites in DATA's zu übernehmen, haben wir sozusagen als kleines Bonbon am Ende dieses Kapitels noch einen kleinen DATA-Wandler abgedruckt. Er läuft sowohl im 64er als auch im 128er-Modus und kann für alle oben beschriebenen Funktionen verwendet werden. Er ist aus Geschwindigkeitsgründen nicht dokumentiert. Doch nun erst einmal die Listings:

```
profi-ass 64 v2.0

110:   c000              .opt p1,oo
120:                ;
130:                ;**********************************
140:                ;* verschieben des zeichensatzes *
150:                ;**********************************
```

```
160:                    ;
170:   c000                     *=   $c000    ; programmstart bei 49152
180:                    ;
190:   d000             vic     =    $d000    ; basisadresse vic
200:   2000             neu     =    8192     ; neuer zeichensatz
210:                    ;
220:   c000 78                  sei           ; interrupt sperren
230:   c001 a5 01               lda  1        ; prozessorport
240:   c003 48                  pha           ; auf stack legen
250:   c004 29 fb               and  #131     ; zeichensatz auswaehlen
260:   c006 85 01               sta  1        ; prozessorport
270:                    ;
280:   c008 a9 00               lda  #<vic    ; low adresse vic
290:   c00a a0 d0               ldy  #>vic    ; high adresse vic
300:   c00c 85 f7               sta  $f7      ; im format low/high
310:   c00e 84 f8               sty  $f8      ; speichern
320:                    ;
330:   c010 a9 00               lda  #<neu    ; adresse neuer zeichensatz
340:   c012 a0 20               ldy  #>neu    ; high-byte der adresse
350:   c014 85 f9               sta  $f9      ; fuer indirekt-indizierte
360:   c016 84 fa               sty  $fa      ; adressierung speichern
370:                    ;
380:   c018 a2 10               ldx  #16      ; 16 mal 256 bytes
390:   c01a a0 00       loop2   ldy  #0       ; y-index auf null setzen
400:   c01c b1 f7       loop1   lda  ($f7),y  ; alten zeichensatz auslesen
410:   c01e 91 f9               sta  ($f9),y  ; und in neuen kopieren
420:   c020 c8                  iny           ; y-index erhoehen
430:   c021 d0 f9               bne  loop1    ; ungleich null dann sprung
440:    c023 e6 f8                 inc  $f8       ; high-byte alte adresse
erhoehen
450:    c025 e6 fa                 inc  $fa       ; high-byte neue adresse
erhoehen
460:   c027 ca                  dex           ; page-zaehler erniedrigen
470:   c028 d0 f0               bne  loop2    ; bei null ende
480:                    ;
490:   c02a 68                  pla           ; alte konfiguration
500:   c02b 85 01               sta  1        ; in prozessorport
510:                    ;
520:   c02d ad 00 dd            lda  56576    ; cia auslesen
530:   c030 09 03               ora  #3       ; bits 1 und 2 setzen
```

```
540:    c032 8d 00 dd         sta  56576    ; und wert zurückschreiben
550:                   ;
560:    c035 ad 18 d0         lda  vic+24   ; register 24 in akku lesen
570:    c038 29 f1            and  #141     ; bits 1-3 loeschen
580:    c03a 09 08            ora  #8       ; und bit drei setzen
590:    c03c 8d 18 d0         sta  vic+24   ; wieder in register 24
600:                   ;
610:    c03f 58               cli           ; interrupt entriegeln
620:    c040 60               rts           ; zurueck zum basic
```

Basiclader:

```
100 rem ************************
110 rem *   rom -> ram copy    *
120 rem *   --basic lader--    *
130 rem ************************
140 :
150 for i=49152 to 49152+65
160 :   read a : rem wert einlesen
170 :   poke i,a : rem in speicher
180 :   s=s+a : rem pruefsumme bilden
190 next i
200 :
210 if s<>8894 then print "fehler in data's !!!" : end
220 :
230 print "programm ok!"
240 :
250 sys 49152 : rem maschinenprogramm aufrufen
260 :
270 data 120,165,  1, 72, 41,251,133
280 data   1,169,  0,160,208,133,247
290 data 132,248,169,  0,160, 32,133
300 data 249,132,250,162, 16,160,  0
310 data 177,247,145,249,200,208,249
320 data 230,248,230,250,202,208,240
330 data 104,133,  1,173,  0,221,  9
340 data   3,141,  0,221,173, 24,208
350 data  41,241,  9,  8,141, 24,208
360 data  88, 96,  0
```

Basicäquivalente:

```
100 rem**********************************
110 rem*    char-copy basicversion     *
120 rem* achtung ! nur im 64er modus *
130 rem**********************************
140 :
150 vic = 53248 : rem basisadresse vic
160 neu = 8192 :  rem neuer zeichensatz
170 cia = 56320 : rem basisadresse cia
175 :
180 poke cia+14,peek (cia+14) and not 1 : rem irq aus
200 poke 1,peek (1) and not 4 : rem charrom auslesen
210 :
220 for i=0 to 4096 : print"",,i
230 :   poke neu+i,peek (vic+i) : rem kopieren
240 next i
250 :
260 poke 1,peek (1) or 4 : rem normal konfiguration
270 poke cia+14,peek (cia+14) or 1 : rem irq frei
280 :
290 poke 56576,peek (56576) or 3 : rem block null
300 t=peek (vic+24) and 241 : rem reg.24 auslesen
310 poke vic+24,t or 2^3 : rem setzen
320 :
330 end : rem programmende
```

Und hier jetzt der DATA-Wandler, den Sie benutzen können um Ihre eigenen Maschinenprogramme in BASIC-Programme einzubauen:

Programmlisting:

```
100 rem ********************
110 rem *   data-wandler   *
120 rem ********************
130 :
```

```
140 input "bereich von";vo
150 input "bereich bis";bi
160 :
170 input "erste zeilennummer";zn
180 input "schrittweite      ";sw
190 :
200 i=0
201 print chr$(147);
210 print zn;"data ";
211 :
220 for j=vo to vo+7
230 :   t=peek (j) : t$=str$ (t)
240 :   t$=right$ (t$,len (t$)-1)
250 :   t$=right$("000"+t$,3)
260 :   print t$;",";
270 next j
271 :
280 print chr$(20)
290 zn=zn+sw
300 vo=vo+8
310 i=i+1
320 if i<7 and vo<bi then goto 210
321 :
330 print "zn=";zn;":sw=";sw;":vo=";vo;":bi=";bi
331 print : print
332 if vo<bi then print "goto 200
340 :     poke 842,19
350 :     for i=843 to 843+9
360 :         poke i,13
370 :     next i
380 :     poke 208,10
390 end
```

2.4.3.3 Modifikation des Zeichensatzes

Da wir nun wissen, wie man einen Zeichensatz kopiert und diesen aktiviert, wollen wir jetzt besprechen, wie man sich neue,

eigene Zeichen erstellt. Dazu muß erst einmal der Aufbau des Zeichensatzes geklärt werden:
Eine Zeichenmatrix besteht aus acht Zeilen mit jeweils acht Punkten, belegt also acht hintereinanderliegende Bytes, deren gesetzte Bits als Punkte in dem Zeichen dargestellt werden. Um nun die Lage eines Zeichens innerhalb des Zeichensatzes zu ermitteln, wendet man folgende Formel an:

Adresse = POKE-Kode * 8

Wir multiplizieren also den Bildschirmkode des Zeichens (Handbuch Anhang) mit acht, da jedes Zeichen ja acht Bytes belegt. Um nun aber die reale Adresse und nicht die relative Adresse zum Beginn des Zeichensatzes zu erhalten, müssen wir zur oben stehenden Formel noch die Basisadresse des Zeichensatzes hinzuaddieren. Die Formel zur Berechnung der realen Speicheradresse eines Zeichens sieht dann im Endeffekt so aus:

Adresse = POKE-Kode * 8 + Basisadresse

Soll auf ein Zeichen im Alternativzeichensatz (Klein/Groß) zugegriffen werden, muß außerdem noch der Wert 2048 ($0800) hinzuaddiert werden, um die 256 Zeichen zu überspringen, die der erste Zeichensatz (Groß/Grafik) belegt. Als Beispiel wollen wir an dieser Stelle einmal das "Sternchen" (ASCII/BSC : 42) in eine Ihnen sicherlich bekannte Spielfigur umdefinieren. Wir gehen dazu folgendermaßen vor: Als erstes kopieren wir mittels der im vorigen Kapitel vorgestellten Maschinenroutine den alten Zeichensatz in den basicspeicher nämlich nach 8192 ($2000). Nach dem Starten dieses Programms liegt der aktuelle Zeichensatz bei 8192 ($2000), so daß wir diesen nur noch so verändern müssen, daß er uns genehm ist. Wollen Sie ein Zeichen allerdings nicht völlig umdefinieren, wie in unserem Beispiel, so wäre es recht praktisch ein Programm zu besitzen, das uns die gewünschte Zeichenmatrix darstellt. Deshalb haben wir für diesen Zweck ein kleines BASIC-Programm geschrieben, daß eigentlich keiner weiteren Erklärung bedürfen sollte. Wollen Sie es auch im 128er-Modus anwenden so müssen Sie die Befehlsfolge in Zeile 310 durch ein GETKEY A$ ersetzen und in der

Zeile 205 noch die Speicherbank mittels des BANK-Befehls definieren.

```
100 rem ********************************
110 rem *   zeichenmatrix anzeigen    *
120 rem ********************************
130 :
140 input "basisadresse zeichensatz";ba
150 :
160 input "bildschirm-zeichen-kode ";zc
170 :
180 input "0. oder 1. zeichensatz  ";zs
190 :
200 ad=ba + zc * 8 + 2048 * zs
210 :
220 for z=0 to 7 : rem zeilen
230 :   for s=0 to 7 : rem spalten
240 :       by=peek (ad+z) and 2^ (7-s)
250 :       a$="+"
260 :       if by<>0 then a$=chr$(18)+" "+chr$(146)
270 :       print a$;
280 :   next s
290 print
295 next z
300 :
310 wait 198,1 : poke 198,0
320 :
330 run
```

Schauen wir uns jetzt doch einmal die Matrix des "Sternchens" an, die wie folgt aussieht:

```
 76543210
0........0 :   0
1.**..**.1 : 102
2..****..2 :  60
3********3 : 255
4..****..4 :  60
5.**..**.5 : 102
6........6 :   0
7........7 :   0
 76543210
```

Ersetzen wir das "Sternchen" nun durch folgende Matrix und schreiben diese mit Hilfe des nachfolgenden Programms in den neuen Zeichensatz:

```
 76543210
0..****..0 :  60
1.******.1 : 126
2*****.**2 : 251
3********3 : 255
4********4 : 255
5****....5 : 240
6.****...6 : 120
7..****..7 : 188
 76543210
```

Programmlisting:

```
100 rem ********************************
110 rem *   neues zeichen (asc:42)     *
120 rem ********************************
130 :
140 ba = 8192 : rem basis neuer zeichensatz
150 cd =   42 : rem zeichenkode
160 sa =    0 : rem gross/grafik satz
170 :
180 ad = ba + cd * 8 + 2048 * sa : rem adresse
190 :
```

```
200 for i=ad to ad+7
210 :   read a :   rem reihe lesen
220 :   poke i,a : rem ablegen
230 next i
240 :
250 end : rem programmende
260 :
270 data  60,126,251,255,255,240,120, 60
```

2.4.3.4 Der Extended-Color-Modus

Wie oben schon erwähnt, existiert noch eine weitere Betriebsart, der sogenannte Extended-Color-Modus (Bit 6 Register 17). Vom Prinzip her ist er dem Multi-Color-Modus sehr ähnlich, nur daß bei ihm die erhöhte Farbauflösung nicht auf Kosten der Punktauflösung, sondern auf Kosten des Zeichenumfangs geht. Ein weiterer Unterschied zum oben beschrieben Multi-Color-Modus besteht darin, daß im Multi-Color-Modus die gesetzten Punkte in einem Kästchen drei verschiedene Farben annehmen können, wärend im Extended-Color-Modus vier verschiedene Hintergrundfarben bei einer Vordergrundfarbe pro Kästchen möglich sind. Wie auch beim Multi-Color-Modus stammen die Farben, um die der normale Textmodus erweitert wurde, wieder aus den Farbregistern 0-3. Wie oben schon erwähnt, geht die erweiterte Farbauflösung auf Kosten des Zeichenumfangs, doch wie genau beeinflußt der Extended-Color-Modus den Zeichenumfang? Da die beiden oberen Bits 6 und 7 des Zeichenkodes auswählen, aus welchem Hintergrundfarbregister die Hintergrundfarbe des Zeichens stammen soll, bleiben nur noch die Bits 0-5 als Zeiger im Charactersatz übrig. Der Zeichenumfang beträgt daher nur noch 2^5=64 Zeichen. Auf welcher Hintergrundfarbe das Zeichen dargestellt wird, können Sie der folgenden Tabelle entnehmen:

00 = Hintergrundfarbregister 0 (Reg.33)
01 = Hintergrundfarbregister 1 (Reg.34)
10 = Hintergrundfarbregister 2 (Reg.35)
11 = Hintergrundfarbregister 3 (Reg.36)

Um nun mittels PRINT Zeichen in einer speziellen Hintergrundfarbe auszugeben, müssen Sie folgende Methode verwenden:

Hintergrundfarbe 0 : print "Zeichen"

Hintergrundfarbe 1 : print "Shift+Zeichen"

Hintergrundfarbe 2 : print "RVSON+Zeichen"

Hintergrundfarbe 3 : print "RVSON+SHift+Zeichen"

Für welche schönen und nützlichen Anwendungen Sie den Extended-Color-Modus alles nutzen können, versucht das folgende Demoprogramm Ihnen einigermaßen zu demonstrieren:

```
100 rem ****************************
110 rem *   extended color demo    *
120 rem ****************************
130 :
140 v=53248 : rem basisadresse vic ii
150 :
160 c (0)=v+04*8 : c(1)=v+05*8
170 c (2)=v+13*8 : c (3)=v+15*8
180 z (0)=068    : z (1)=133
190 z (2)=205    : z (3)=079
200 :
210 poke53280,0 : poke53281,0 : print chr$(147)
220 :
230 pokev+17,peek (v+17) or 64
240 :
250 poke v+34,6
260 poke v+35,7
270 poke v+36,2
```

```
280 :
290 :
300 for i=55296 to 56295 : poke i,1 : next i
310 :
320 poke 56320+14,peek (56320+14) and 254
330 poke 1,peek (1) and not 4
340 print chr$(147);
350 :
360 for x=0 to 3
370 :    for y=0 to 7
380 :         for z=0 to 7
390 :
400 :              t=peek (c (x)+y) and (2^z)
410 :
420 :              if t=0 then 490
430 :
440 :              ad=11-z+(x*8)+(4+y)*40
450 :
460 :              poke  1024+ad,z (x)
470 :              poke 55296+ad,1
480 :
490 :         next z
500 :    next y
510 next x
520 poke 1,peek (1) or 4
530 poke 56320+14,peek (56320+14) or 1
540 :
550 for i=1 to 500
560 :
570 :    t=peek (v+36)
580 :    poke v+36,peek (v+35)
590 :    poke v+35,peek (v+34)
600 :    poke v+34,t
610 :
620 :    for pa=1 to 50:next pa
630 :
640 next i
```

2.4.3.5 Der Multi-Color-Modus

Nachdem Sie nun wissen, wie Sie eigene Zeichen kreieren können, wollen wir Ihnen noch einige zusätzliche Features des VIC's erklären. Es sind dies der Multi-Color- und der weiter unten beschriebene Extended-Color-Modus. In diesen beiden Betriebsmodi ist es möglich, in einem Zeichen mehr als eine Farbe zu verwenden. Im Multi-Color-Modus kann ein Zeichen aus maximal vier verschiedenen Farben bestehen. Drei dieser Farben sind für alle Zeichen gleich und stammen aus den Hintergrundfarbregistern 0-2. Doch wie verwaltet der Videocontroller den Multi-Color-Modus? Nun, die erhöhte Farbauflösung geht wieder einmal auf Kosten der Punktauflösung. Ein Zeichen besteht nämlich im Multi-Color-Modus nicht mehr aus einer Matrix von 8*8 Punkten, sondern nur noch aus einer Matrix von 4*8 Punkten. Es kodieren jetzt jeweils zwei nebeneinanderliegende Bitpärchen einen Punkt, wobei sie zusätzlich bestimmen, woher diese Farbe stammen soll. Hier nun die den einzelnen Bitkombinationen zugeordneten Farbquellen:

00 = Hintergrundfarbregister 0 (Reg.33)
01 = Hintergrundfarbregister 1 (Reg.34)
10 = Hintergrundfarbregister 2 (Reg.35)
11 = Drei Bits des Farb-RAMs.

Wie ihnen aufgefallen sein sollte, wird die Farbe, wenn beide Bits gesetzt sind, nicht mehr aus allen vier Bits gebildet sondern nur noch aus drei Bits. Das vierte Bit dient nämlich zur Auswahl, ob das Zeichen als Multicolorzeichen oder weiterhin als normales Zeichen dargestellt werden soll. Mit diesen Kombinationen ist es z. B. möglich, hochauflösende Farbbilder zu erstellen und gleichzeitig noch normale Schrift anzuzeigen. Diese Strategie benutzt ein Großteil der zu kaufenden Spiele. Sie sehen, der veränderte Zeichensatz stellt zusammen mit dem Multi-Color-Modus eine lohnende Alternative zur HI-RES Grafik dar, zumal die Bearbeitungsgeschwindigkeit um einiges höher ist. Als Abschluß dieser Erklärung zum Multi-Color-Modus, hier noch ein kleines Demoprogramm, daß Ihnen den

Effekt demonstriert, den Sie durch Mischung von normalen und Multicolorzeichen erreichen können. Es läuft wie auch alle anderen Programme im 64er Modus, obwohl es auch im 128er Modus ohne Probleme laufen würde, da das Multicolorflag nicht vom Interpreter beeinflusst wird, solange der Textmodus aktiv ist. Hier nun das Demonstrationsprogramm:

Programmlisting:

```
100 rem ******************************
110 rem *   Multi-Color-Modus demo *
120 rem ******************************
130 :
140 poke 53280,0 : poke 53281,0 : rem hintergrund und rahmen
150 :
160 v=53248 : rem basisadresse vic
170 :
180 poke v+22,peek (v+22) or 2^4 : rem Multi-Color-Modus
190 :
200 poke v+34,2 : rem multicolorfarbe 0
210 poke v+35,7 : rem multicolorfarbe 1
220 :
230 for i=0 to 999
240 :   poke 1024+i,i and 255
250 :   poke 55296+i,i and 15
260 next i
270 :
280 goto 280 : rem programmende
```

2.4.3.6 Chardesigner (40-Zeichenmodus)

Nachdem wir den Textmodus in Verwendung mit einem neuen Zeichensatz in den höchsten Tönen gelobt haben, wollen wir Ihnen an dieser Stelle ein Programm vorstellen, mit dem die Kreation eines neuen Zeichensatzes wesentlich vereinfacht wird. Es ist dies der sogenannte Charactergenerator. Er bietet folgende

Möglichkeiten bei der Zeichenerstellung: Punkt setzen, Punkt löschen, an der Senkrechten spiegeln, an der horizontalen spiegeln, Rotieren, Scrollen (Oben, Unten, Links, Rechts), Invertieren, Kopieren, Übernehmen, Löschen, Laden, Speichern, Farbwahl und noch einiges mehr.

Im wesentlichen ist er in drei Feldern aufgebaut:

1. Das Erstellfeld. In diesem Feld laufen fast alle oben beschriebenen Funktionen ab, wie z.B. Scrollen, Rotieren, Punkt setzen u.s.w. Es dient im wesentlichen aber zum Erstellen der Zeichenmatrix, wobei das Feld noch einmal in verkleinerter Form weiter oben dargestellt wird. Hierdurch haben Sie immer einen Vergleich, wie das Zeichen in Realgröße aussehen würde.

2. Das Kopiefeld für die bereits neu erstellten Zeichen. Die unter Feld 2 stehende Matrix können Sie mittels des Pfeils nach oben in das Kopiefeld übernehmen. Dieses Feld soll als Vergleichsmöglichkeit zu bereits vorher erstellten Zeichen dienen und kann auch in das Erstellfeld übernommen werden.

3. Das zweite Kopiefeld. Prinzipiell sind die Funktionen der beiden Kopiefelder identisch, nur das im zweiten Kopiefeld immer die Matrix des Orginalzeichensatzes (ROM) steht.

Nun, da wir über den grundsätzlichen Aufbau Bescheid wissen, wollen wir näher auf die einzelnen Befehle eingehen, die alle durch das Drücken der speziellen Tasten ausgelöst werden. Eine genaue Programmerklärung, wie in den Statistikkapiteln, soll an dieser Stelle nicht folgen, da das Programm so komplex ist, daß die Beschreibung zu viele Seiten in Anspruch nehmen würde. Eine etwas allgemeiner gehaltene Programmbeschreibung ist natürlich wieder vorhanden und steht wie immer am Ende dieses Kapitels.

Befehlserklärung:

Cursor Rechts: Wird diese Taste gedrückt, so wird der Zeichencursor im Feld 1 um eine Position nach rechts bewegt, sollte er dabei den Rand überschreiten, so wird er auf die erste Spalte der gleichen Zeile gesetzt.

Cursor Links: Die Funktion stellt das Gegenteil zur Cursor Rechts Funktion dar und bewegt den Zeichencursor um eine Position nach links. Wird hierbei der Rand überschritten, so wird der Cursor auf die letzte Spalte der gleichen Reihe gesetzt.

Cursor Oben: Der Zeichencursor wird um eine Zeile nach oben bewegt. Überschreitet er dabei den Rand, so wird er auf die gleiche Spalte der untersten Zeile gesetzt.

Cursor Unten: Wird diese Taste gedrückt, so wird der Cursor um eine Zeile nach unten bewegt. Überschreitet er hierbei den Rand, so wird er in der gleichenSpalte auf die oberste Reihe gesetzt.

, und .: Erniedrigt oder erhöht die Codenummer die im Bereich von 0 bist 511 liegt. Liegt die Codenummer über 255, so wird die Matrix dem Klein/Groß Zeichensatz entnommen.

< und >: Diese Tasten haben dieselbe Funktion, wie die beiden letzten beschriebenen Tasten, nur daß die Codenummer um 10 erniedrigt oder erhöht wird. Hiermit ist ein schnellers Aufsuchen des zu verändernden Codes möglich.

Space: Wir die Space-Taste gedrückt, so wird an der Position des Zeichencursors ein Punkt gesetzt und die verkleinerte Matrix aktualisiert. Der Zeichencursor wird nach Betätigung der Spacetaste nicht um eine Spalte weiterbewegt!

Shift Space: Die Tastenkombination Shift Space stellt das Gegenteil zur Space Taste dar, dient also zum Löschen eines Punktes im Erstellfeld.

V: Wird die V-Taste gedrückt, so wird das Erstellfeld an der vertikalen Achse gespiegelt und die verkleinerte Zeichenmatrix aktualisiert.

H: Auch diese Funktion dient zum Spiegeln der Zeichenmatrix, nur daß die Spiegelung diesmal nicht an der vertikalen Achse, sondern an der horizontalen Achse vorgenommen wird.

R: Mit dieser Taste können Sie eine weitere nützliche Funktion des Charactergenerators ansprechen. Wird nämlich diese Taste gedrückt, so wird das Zeichen um 90 Grad im Uhrzeigersinn rotiert. Soll das Zeichen im Gegenuhrzeigersinn rotiert werden, so ist diese Funktion dreimal hintereinander auszuführen

S: Scrollen. Nachdem Sie diese Taste gedrückt haben, ändert sich die Rahmenfarbe des Bildschirms von Schwarz auf Rot. Dies signalisiert Ihnen, daß Sie sich nicht in der normalen Tastaturabfrage befinden, sondern eine spezielle Eingabe von Ihnen gefordert wird. Sie können jetzt 'L' für Links,'R' für Rechts, 'O' für Oben und 'U' für Unten eingeben. Je nach der gedrückten Taste wird das Zeichen dann in die angegebene Richtung gescrollt. Haben Sie eine der vier erlaubten Tasten gedrückt, so ändert sich die Rahmenfarbe von Rot nach Grün, um Ihnen anzuzeigen, daß Ihre Eingabe korrekt war. Andernfalls wird der Rahmen wieder auf Schwarz gestellt und gar keine Verschiebung ausgeführt.

I: Mittels dieser Taste können Sie das Erstellfeld invertieren (negieren). Dies können Sie zum Beispiel bei der Erstellung eines ausländischen Zeichensatzes verwenden, der auch die Sonderzeichen in reverser Form enthalten soll.

Pfeil n. oben: Mit Hilfe dieser Taste ist es möglich die verkleinerten Zeichen der Felder 2 und 3 in vergrößerter Form in den beiden Kopierfeldern abzubilden. Dies ist zum Beispiel dann notwendig, wenn Sie eines dieser Felder in das Erstellfeld übernehmen wollen.

2: Drücken Sie zwei, so wird das erste Kopierfeld (2) in das Eingabefeld kopiert. Hierzu muß das gewünschte Zeichen erst mittels des Pfeils nach oben in die beiden Kopiefelder kopiert worden sein.

3: Hiermit lösen Sie die gleiche Funktion aus, wie mit der '2' , nur daß alle Angaben sich jetzt auf das zweite Kopiefeld (3) beziehen.

Return: Haben Sie ein Zeichen fertig erstellt und wollen dieses in den neuen Zeichensatz übernehmen, so müssen Sie die Return-Taste betätigen. Diese kopiert dann das Erstellfeld in den neuen Zeichensatz, den Sie dan später abspeichern können.

Clr: Löscht das Erstellfeld.

Home: Führt ein Cursor-Home im Eingabefenster durch.

L: Lädt einen Charactersatz von der Floppy. Der Name wird am unteren Bildschirmrand erwartet.

A: Diese Funktion stellt das Gegenteil zur 'L'-Funktion dar und speichert einen Zeichensatz auf der Floppy ab.

+ und -: Hiermit erhöhen oder erniedrigen Sie die aktuelle Zeichenfarbe. Diese wird dann im Feld 'Farb-RAM' dargestellt. Das Eingabefeld wird hierbei auch berichtigt.

Shift + und-: Betätigen Sie eine der beiden Tastenkombinationen, so erhöhen oder erniedrigen Sie die Hintergrundfarbe.

: und ;: Diese Funktionen haben eigentlich nichts mit der Zeichenerstellung zu tun und dienen dazu die Zeichencursorfarbe entweder zu erhöhen oder zu erniedrigen.

Nachdem wir Ihnen bis hierhin nun alle Befehle vorgestellt haben, soll jetzt das BASIC-Listing folgen, damit Sie auch alle Befehle einmal ausprobieren können. Das Programm läuft diesmal im 128er Modus. Eine Umschaltung auf den neuen Zeichensatz ist im 128er-Modus recht einfach. Sie brauchen nur folgende Befehlssequenz eingeben und schon ist Ihr neuer Zeichensatz installiert:

```
GRAPHIC 1
BLOAD "Programmname",8 : GRAPHIC 0
POKE 2604,24
```

Das Register 2604 der sogenannten Zeropage entspricht im Aufbau und in der Funktion dem VIC-Register 24 und definiert die Basisadresse des Zeichensatzes im Textmodus.
Wollen Sie den neuen Zeichensatz allerdings im 64er Modus nutzen, so ist dies schon etwas problematischer. Sie müssen dazu folgendermaßen vorgehen:

POKE 43,1 : POKE 44,64
POKE 16384,0 : NEW
LOAD "Programmname",8,1 : POKE 53248,24

Im Gegensatz zur ersten Methode funktioniert diese Befehlsfolge nur im Direktmodus, also nicht innerhalb eines Programms.

Hier aber jetzt, wie versprochen, das Programmlisting. Lassen Sie sich bitte nicht durch die Länge schockieren, das Abtippen lohnt sich bestimmt.

Basiclisting:

```
100 REM ********************************
110 REM * --- CHARACTERGENERATOR --- *
120 REM *  FUER DEN 40-ZEICHENMODUS  *
130 REM ********************************
140 :
150 C1= 6 : C2=1 : CC=0 : REM EINSCHALTFARBEN
160 :
170 COLOR 0,1 : COLOR 4,1 : COLOR 5,15 : GRAPHIC 1 : GRAPHIC 0
180 :
190 : FOR I=2816 TO 2904
200 :      READ A : REM DATA'S EINLESEN
210 :      S=S+A : REM PRUEFSUMME BILDEN
220 :      POKE I,A : REM SPEICHERN
230 : NEXT I
240 :
250 IF S<>11125 THEN PRINT "FEHLER IN DATA'S !!!" : END
260 :
270 PRINT "PROGRAMM OK!"
280 :
290 GRAPHIC 1,1 : REM GRAFIK EINSCHALTEN UND LOESCHEN
300 :
310 CHAR 1,0,0,"#" : REM CURSOR AUSGEBEN
320 SSHAPE C$,0,0,23,20 : REM CURSOR IN C$ SPEICHERN
330 SPRSAV C$,1 : REM IN BLOCK FUER SPRITE 1 KOPIEREN
340 :
350 SSHAPE C$,40,0,63,21 : REM LEERSTRING EINLESEN
360 SPRSAV C$,2 : REM SPRITE 2 LOESCHEN
370 SPRSAV C$,3 : REM SPRITE 3 LOESCHEN
380 SPRSAV C$,4 : REM SPRITE 4 LOESCHEN
390 :
400 BOX 1,0,0,23,20,0,1 : REM AUSGEFUELLTES RECHTECK
410 SSHAPE C$,0,0,23,20 : REM ALS SHAPE SPEICHERN
420 SPRSAV C$,5 : REM UND ALS MATRIX FUER DIE SPRITES
430 SPRSAV C$,6 : REM FUENF, SECHS, SIEBEN
440 SPRSAV C$,7 : REM UND ACHT SETZEN
450 SPRSAV C$,8
460 :
470 FAST : REM FAST-MODUS
480 : FOR I=0 TO 4095
490 :      BANK 14 : REM CHARACTERROM AUSWAEHLEN
500 :      T=PEEK(53248+I) : REM BYTE LESEN
510 :      BANK 0 : REM UND WIEDER BANK NULL SELEKTIEREN
520 :      POKE 8192+I,T : REM BYTE IN KOPIE SCHREIBEN
530 : NEXT I
540 SLOW : REM SLOW-MODUS
550 :
560 GRAPHIC 0 : REM UND GRAFIK WIEDER AUSSCHALTEN
570 :
```

```
580 SPRITE 1,1,CC+1,0,0,0,0 : REM PARAMETER
590 SPRITE 2,1,C2+1,0,1,1,0 : REM FUER ALLE
600 SPRITE 3,1,C2+1,0,1,1,0 : REM ACHT SPRITES
610 SPRITE 4,1,C2+1,0,1,1,0 : REM SETZEN
620 SPRITE 5,1,C1+1,1,1,1,0
630 SPRITE 6,1,C1+1,1,1,1,0
640 SPRITE 7,1,C1+1,1,1,1,0
650 SPRITE 8,1,C1+1,1,1,1,0
660 :
670 MOVSPR 1, 48,146 : REM CURSOR
680 MOVSPR 2,208,106 : REM FELD ZWEI
690 MOVSPR 3,304,106 : REM FELD DREI
700 MOVSPR 4,104,106 : REM FELD EINS
710 MOVSPR 5, 48,146 : REM HINTERGRUND
720 MOVSPR 6, 64,146 : REM VON FELD EINS
730 MOVSPR 7, 48,168
740 MOVSPR 8, 64,168
750 :
760 EC$=CHR$ (27) : REM ESCAPE
770 DIM A(7,7) : REM SPEICHER RESERVIEREN FUER ROTIEREN
780 :
790 REM ************************
800 REM *   BILDSCHIRMAUFBAU   *
810 REM ************************
820 :
830 PRINT CHR$(19);CHR$(19) : REM ALLE WINDOWS LOESCHEN
840 SCNCLR : REM BILDSCHIRM LOESCHEN
850 :
860 PRINT "┌──────────────────┐
870 PRINT "│ ZEICHENKODE: 000 │
880 PRINT "└──────────────────┘
890 PRINT "┌────────────────┬─────────────────────┐
900 PRINT "│ FARBRAM:       │ HINTERGRUND:        │
910 PRINT "└────────────────┴─────────────────────┘
920 PRINT "┌────────────────┬────────────────┬────────────────┐";
930 PRINT "│ FELD 1:        │ FELD 1:        │ FELD 3:        │";
940 PRINT "│                │                │                │";
950 PRINT "└────────────────┴────────────────┴────────────────┘";
960 :
970 PRINT "   01234567        01234567        01234567
980 PRINT "   ┌────────┐      ┌────────┐      ┌────────┐
990 PRINT " 0│          │0 0│          │0 0│          │0
1000 PRINT " 1│          │1 1│          │1 1│          │1
1010 PRINT " 2│          │2 2│          │2 2│          │2
1020 PRINT " 3│          │3 3│          │3 3│          │3
1030 PRINT " 4│          │4 4│          │4 4│          │4
1040 PRINT " 5│          │5 5│          │5 5│          │5
1050 PRINT " 6│          │6 6│          │6 6│          │6
1060 PRINT " 7│          │7 7│          │7 7│          │7
1070 PRINT "   └────────┘      └────────┘      └────────┘
1080 PRINT "   01234567        01234567        01234567
1090 :
1100 GOTO 3690
1110 :
1120 WINDOW 3,12,11,19 : PRINT EC$;"M" : REM WINDOW = FELD 1
1130 :
1140 X=0 : Y=0 : REM CURSOR OBEN LINKS
1150 :
1160 REM ***********************
1170 REM *   TASTATURABFRAGE   *
1180 REM ***********************
1190 :
1200 MOVSPR 1,48+8*X,146+8*Y : REM CURSOR BEWEGEN
1210 SYS 2816 : REM FELD 1 ALS SPRITE DARSTELLEN
1220 :
1230 GETKEY A$ : REM TASTATUR ABFRAGEN
1240 :
```

```
1250 IF A$="▐]" THEN X=X+1 : X=X AND 7 : REM CURSOR RECHTS
1260 IF A$="▐▌" THEN X=X-1 : X=X AND 7 : REM CURSOR LINKS
1270 IF A$="▣" THEN Y=Y+1 : Y=Y AND 7 : REM CURSOR OBEN
1280 IF A$="⊐" THEN Y=Y-1 : Y=Y AND 7 : REM CURSOR UNTEN
1290 :
1300 IF A$="." THEN CD=(CD+1) AND 511 : REM CDNR MINUS 1
1310 IF A$="," THEN CD=(CD-1) AND 511 : REM CDNR PLUS 1
1320 IF A$=">" THEN CD=(CD+10) AND 511 : REM CDNR PLUS 10
1330 IF A$="<" THEN CD=(CD-10) AND 511 : REM CDNR MINUS 10
1340 :
1350 IF A$="," OR A$="." THEN 4050 : REM CDNR AUSGEBEN
1360 IF A$="<" OR A$=">" THEN 4050 : REM CDNR AUSGEBEN
1370 :
1380 IF A$="▐]" OR A$="▐▌" THEN 1200 : REM ZUM ANFANG
1390 IF A$="▣" OR A$="⊐" THEN 1200
1400 :
1410 IF A$=" " THEN CHAR 1,X,Y,"▨ " : GOTO 1200 : REM PUNKT SETZEN
1420 IF A$=CHR$(160) THEN CHAR 1,X,Y,"■ " : GOTO 1200 : REM PUNKT LOESCHEN
1430 :
1440 IF A$="V" THEN 1960 : REM SPIEGELN
1450 IF A$="H" THEN 2130 : REM SPIEGELN
1460 IF A$="R" THEN 2300 : REM ROTIEREN
1470 IF A$="S" THEN 2560 : REM SCROLLEN
1480 IF A$="I" THEN 1820 : REM INVERT.
1490 IF A$="↑" THEN 4200 : REM ZEIGEN
1500 IF A$="←" THEN 4200 : REM ZEIGEN
1510 :
1520 IF A$="2" OR A$="3" THEN 4720 : REM KOPIEREN
1530 :
1540 IF A$=CHR$(13) THEN 4570 : REM UEBERNEHMEN
1550 :
1560 IF A$="⊐" THEN BEGIN : REM WINDOW LOESCHEN
1570 :    WINDOW 3,12,10,19 : REM NEU DEFINIEREN
1580 :    SCNCLR : REM LOESCHEN
1590 :    GOTO 1200
1600 BEND : REM WEITER
1610 :
1620 IF A$="▣" THEN X=0 : Y=0 : REM HOME IM FENSTER
1630 :
1640 IF A$="L" THEN 3480 : REM LADEN
1650 IF A$="A" THEN 3590 : REM SAVEN
1660 :
1670 IF A$="+" THEN C2=(C2+1) AND 15 : REM FARBR. +1
1680 IF A$="-" THEN C2=(C2-1) AND 15 : REM FARBR. -1
1690 IF A$="┼" THEN C1=(C1+1) AND 15 : REM HINTERGR. +1
1700 IF A$="|" THEN C1=(C1-1) AND 15 : REM HINTERGR. -1
1710 IF A$="+" OR A$="-" THEN 3700 : REM FARBEN
1720 IF A$="┼" OR A$="|" THEN 3700 : REM SETZEN
1730 :
1740 IF A$=":" THEN CC=(CC+1) AND 15 : REM CURSORFARBE
1750 IF A$=";" THEN CC=(CC-1) AND 15 : REM AENDERN
1760 :
1770 SPRITE 1,1,CC+1 : REM CURSORFARBE AENDERN
1780 :
1790 GOTO 1230 : REM ZUM ANFANG
1800 :
1810 REM ******************
1820 REM *   INVERTIEREN    *
1830 REM ******************
1840 :
1850 FAST : REM 2-MHZ SYSTEMTAKT
1860 : FOR I=483 TO 763 STEP 40 : REM FELD 1
1870 :      FOR J=I TO I+7 : REM INVERTIEREN
1880 :           POKE 1024+J,XOR (PEEK (1024+J),128)
1890 :      NEXT J
1900 : NEXT I
```

```
1910 SLOW : REM 1-MHZ SYSTEMTAKT
1920 :
1930 GOTO 1200
1940 :
1950 REM **************************
1960 REM *   VERTIKAL SPIEGELN    *
1970 REM **************************
1980 :
1990 FAST : REM FAST-MODUS
2000 : FOR I=1507 TO 1787 STEP 40
2010 :      FOR J=0 TO 3
2020 :           A1=PEEK (I+J) : REM WERTE EINLESEN
2030 :           A2=PEEK (I+7-J)
2040 :           POKE I+J,A2 :   REM UND VERTAUSCHEN
2050 :           POKE I+7-J,A1
2060 :      NEXT J
2070 NEXT I
2080 SLOW : REM SLOW-MODUS
2090 :
2100 GOTO 1200 : REM ZURUECK
2110 :
2120 REM ****************************
2130 REM *   HORIZONTAL SPIEGELN    *
2140 REM ****************************
2150 :
2160 FAST : REM SYSTEMTAKT ERHOEHEN
2170 : FOR I=1507 TO 1514 : REM SPALTEN
2180 :      FOR J=0 TO 120 STEP 40 : REM ZEILEN
2190 :           A1=PEEK (I+J) : REM WERTE EINLESEN
2200 :           A2=PEEK (I+280-J)
2210 :           POKE I+J,A2 : REM UND DANN UMDREHEN
2220 :           POKE I+280-J,A1
2230 :      NEXT J
2240 : NEXT I
2250 SLOW : REM WIEDER SLOW-MODUS
2260 :
2270 GOTO 1200
2280 :
2290 REM ****************
2300 REM *   ROTIEREN   *
2310 REM ****************
2320 :
2330 FAST : REM SCHNELL-MODUS
2340 : Z=0 : REM INDEX FUER ARRAY
2350 : FOR I=1507 TO 1787 STEP 40 : REM ZEILEN
2360 :      FOR J=0 TO 7 : REM SPALTEN
2370 :           A(Z,J)=PEEK (I+J) : REM WERT IM ARRAY
2380 :      NEXT J :                REM SPEICHERN
2390 :      Z=Z+1 : REM INDEX ERHOEHEN
2400 : NEXT I
2410 :
2420 : Z1=0 : REM 1. INDEX AUF NULL
2430 : FOR I=1514 TO 1507 STEP -1 : REM SPALTEN
2440 :      Z2=0 : REM 2. INDEX AUF NULL
2450 :      FOR J=0 TO 280 STEP 40 : REM WERTE VERSETZT
2460 :           POKE I+J,A(Z1,Z2) : REM WIEDER EINLESEN
2470 :           Z2=Z2+1 : REM 2. INDEX ERHOEHEN
2480 :      NEXT J
2490 :      Z1=Z1+1 : REM 1. INDEX ERHOEHEN
2500 : NEXT I
2510 SLOW : REM WIEDER IN SLOW-MODUS GEHEN
2520 :
2530 GOTO 1200 : REM YURUECK
2540 :
2550 REM *******************
2560 REM *   VERSCHIEBEN   *
2570 REM *******************
```

```
2580 :
2590 COLOR 4,3 : REM RAMEN ROT FAERBEN
2600 :
2610 GETKEY A$ : REM TASTE HOLEN
2620 :
2630 IF A$=CHR$(13) THEN BEGIN : REM TASTE = RETURNCLOSE?
2640 :     COLOR 4,1 : REM JA, DANN RAMEN WIEDER SCHWARZ
2650 :     GOTO 1230 : REM UND RUECKSPRUNG
2660 BEND : REM THEN-ANWEISUNG ABSCHLIESSEN
2670 :
2680 FAST : REM FAST-MODUS AKTIVIEREN
2690 : COLOR 4,6 : REM RAMEN GRUEN FUER ALLES OK
2700 :
2710 : WINDOW 3,12,10,19 : REM WINDOW NEU DEFINIEREN
2720 :
2730 : IF A$="L" THEN BEGIN : REM LINKS ROLLEN ?
2740 :       Z=0 : REM ARRAY-INDEX AUF NULL SETZEN
2750 :       FOR I=1507 TO 1787 STEP 40 : REM ZEILEN
2760 :           A(Z,1)=PEEK (I) : REM LINKE SPALTE SPEICHERN
2770 :           Z=Z+1 : REM INDEX ERHOEHEN
2780 :       NEXT I
2790 :
2800 :       FOR I=0 TO 7 : REM ERSTE SPALTE
2810 :           PRINT "■";CHR$ (20) : REM LOESCHEN
2820 :       NEXT I
2830 :
2840 :       Z=0
2850 :       FOR I=1514 TO 1794 STEP 40 : REM GESPEICHERTE
2860 :           POKE I,A(Z,1) : REM ZEILE RECHTS
2870 :           Z=Z+1 : REM ANFUEGEN
2880 :       NEXT I
2890 : BEND : REM ENDE DES LINKS-SCROLLENS
2900 :
2910 : IF A$="R" THEN BEGIN
2920 :       Z=0 : REM INDEX AUF NULL SETZEN
2930 :       FOR I=1514 TO 1794 STEP 40 : REM ZEILEN
2940 :           A(Z,1)=PEEK (I) : REM RECHTE SPALTE
2950 :           POKE I,32 : REM SPEICHERN UND LOESCHEN
2960 :           Z=Z+1 : REM INDEX ERHOEHEN
2970 :       NEXT I
2980 :
2990 :       FOR I=0 TO 7 : REM NACH RECHTS
3000 :           PRINT CHR$(148) : REM SCROLLEN
3010 :       NEXT I
3020 :
3030 :       Z=0 : REM INDEX AUF NULL
3040 :       FOR I=1507 TO 1787 STEP 40 : REM GESPEICHERTEN
3050 :           POKE I,A(Z,1) : REM WERTE ZURUECKSCHREIBEN
3060 :           Z=Z+1 : REM INDEX WIEDER ERHOEHEN
3070 :       NEXT I
3080 : BEND : REM ENDE DER 'THEN'-ANWEISUNGEN
3090 :
3100 : IF A$="O" THEN BEGIN : REM NACH OBEN SCROLLEN
3110 :       Z=0 : REM ARRAY-INDEX AUF NULL SETZEN
3120 :       FOR I=1507 TO 1514 : REM OBERSTE ZEILE
3130 :           A(Z,1)=PEEK (I) : REM SPEICHERN
3140 :           Z=Z+1 : REM INDEX ERHOEHEN
3150 :           POKE I,32 : REM UND ZEILE LOESCHEN
3160 :       NEXT I
3170 :
3180 :       PRINT EC$;"V" : REM WINDOW NACH OBEN SCROLLEN
3190 :
3200 :       Z=0 : REM INDEX AUF NULL SETZEN
3210 :       FOR I=1787 TO 1794 : REM GESPEICHERTE ZEILE
3220 :           POKE I,A(Z,1) : REM UNTEN ANFUEGN
3230 :           Z=Z+1 : REM INDEX ERHOEHEN
3240 :       NEXT I
```

```
3250 : BEND : REM ENDE DER TEILROUTINE
3260 :
3270 : IF A$="U" THEN BEGIN : REM NACH UNTEN SCROLLEN
3280 :       Z=Ø : REM FELD-INDEX AUF NULL SETZEN
3290 :       FOR I=1787 TO 1794 : REM UNTERSTE ZEILE
3300 :           A(Z,1)=PEEK (I) : REM ZWISCHENSPEICHERN
3310 :           Z=Z+1 : REM INDEX ERHOEHEN
3320 :           POKE I,32 : REM ZEILE LOSCHEN
3330 :       NEXT I
3340 :
3350 :       PRINT EC$;"W" : REM WINDOW NACH UNTEN SCROLEN
3360 :
3370 :       Z=Ø : REM ZWISCHENSPEICHERINDEX AUF NULL
3380 :       FOR I=15Ø7 TO 1514 : REM GESPEICHERTE ZEILE
3390 :           POKE I,A(Z,1) : REM OBEN ANFUEGEN
3400 :           Z=Z+1 : REM INDEX ERHOEHEN
3410 :       NEXT I
3420 : BEND : REM ENDE DER UNTERROUTINE
3430 SLOW : REM IN SLOW-MODUS ZURUECKSCHALTEN
3440 :
3450 COLOR 4,1 : GOTO 12ØØ : REM SCHWARZER RAMEN
3460 :
3470 REM *************************
3480 REM *   ZEICHENSATZ LADEN   *
3490 REM *************************
3500 :
3510 PRINT CHR$(19);CHR$(19)
3520 CHAR 1,Ø,24,"PROGRAMMNAME"
3530 WINDOW 15,24,15+18,24
3540 INPUT NA$ : REM NAMEN EINLESEN
3550 BLOAD (NA$),BØ : REM ZEICHENSATZ LADEN
3560 GOTO 112Ø : REM ZURUECK
3570 :
3580 REM ******************************
3590 REM *   ZEICHENSATZ SPEICHERN    *
3600 REM ******************************
3610 :
3620 PRINT CHR$(19);CHR$(19) : REM WINDOW LOESCHEN
3630 CHAR 1,Ø,24,"PROGRAMMNAME"
3640 WINDOW 15,24,15+18,24
3650 INPUT NA$ : REM PROGRAMMNAMEN EINLESEN
3660 BSAVE (NA$),BØ,P8192 TO P12288
3670 GOTO 112Ø : REM ZURUECK
3680 :
3690 REM ***********************
3700 REM *   FARBEN ANZEIGEN   *
3710 REM ***********************
3720 :
3730 FAST : REM FAST-MODUS
3740 : PRINT CHR$(19);CHR$(19)
3750 : COLOR 5,C1+1 : REM ZEICHENFARBE GLEICH HINTERGRUND
3760 : CHAR 1,3Ø,4,"▣  ■" : REM FARBBALKEN
3770 : CHAR 1,1Ø,7,"▣  ■" : REM AUSGEBEN
3780 : CHAR 1,1Ø,8,"▣  ■" : REM HINTERGRUNDFARBEN
3790 : CHAR 1,23,7,"▣  ■" : REM FUER SPRITES
3800 : CHAR 1,23,8,"▣  ■" : REM AUSGEBEN
3810 : CHAR 1,35,7,"▣  ■"
3820 : CHAR 1,35,8,"▣  ■"
3830 :
3840 : COLOR 5,C2+1 : REM ZEICHENFARBE GLEICH SPRITEFARBE
3850 : CHAR 1,11,4,"▣  ■" : REM FARBBALKEN AUSGEBEN
3860 : BANK 15 : REM I/O-BEREICH AUSWAEHLEN
3870 : FOR I=55779 TO 56Ø59 STEP 4Ø : REM FARBE VON FELD
3880 :     FOR J=I TO I+7 : REM EINS AKTUALISIEREN
3890 :         POKE J,C2
3900 :     NEXT J
3910 : NEXT I
```

```
3920 SLOW : REM SLOW-MODUS
3930 :
3940 SPRITE 2,1,C2+1 : REM FARBEN DER VERGROESSERTEN
3950 SPRITE 3,1,C2+1 : REM ZEICHEN ANPASSEN
3960 SPRITE 4,1,C2+1
3970 :
3980 SPRITE 5,1,C1+1 : REM HINTERGRUNDFARBE VON
3990 SPRITE 6,1,C1+1 : REM FELD EINS
4000 SPRITE 7,1,C1+1 : REM KORRIGIEREN
4010 SPRITE 8,1,C1+1
4020 :
4030 GOTO 1120 : REM ZUR TASTATURABFRAGE
4040 :
4050 REM **************************
4060 REM *   CODENUMMER AENDERN   *
4070 REM **************************
4080 :
4090 FAST : REM FAST-MODUS AKTIVIEREN
4100 : PRINT CHR$(19);CHR$(19)
4110 : Z$=STR$(CD) : REM CODENUMMER IN STRING WANDELN
4120 : Z$=RIGHT$(Z$,LEN(Z$)-1) : REM STRING FORMATIEREN
4130 : Z$=RIGHT$("00"+Z$,3) : REM DREI ZEICHEN LAENGE
4140 : COLOR 5,15 : REM ZEICHENFARBE IST HELLBLAU
4150 : PRINT SPC(15);Z$ : REM ZAHLENSTRING AUSGEBEN
4160 : COLOR 5,C2+1 : REM FARBE WIEDER AKTUALISIEREN
4170 : A$="←" : REM NUR SPRITES ANZEIGEN
4180 :
4190 REM **************************
4200 REM *   ZEICHEN DARSTELLEN   *
4210 REM **************************
4220 :
4230 A1=53248+CD*8 : A2=8192+CD*8 : REM ADRESSEN ZEICHENS.
4240 FAST : REM FAST-BETRIEBSART
4250 : Z=0 : REM SPRITE-ZEIGER AUF NULL
4260 : FOR I=0 TO 7
4270 :      BANK14:POKE 57*64+Z,PEEK(A1+I) : REM ALTES ZEICHEN
4280 :      BANK 0:POKE 58*64+Z,PEEK(A2+I) : REM NEUES ZEICHEN
4290 : Z=Z+3 : REM SPRITE-ZEIGER AUF NAECHSTE ZEILE SETZEN
4300 : NEXT I
4310 :
4320 : IF A$="←" THEN 4530 : REM NUR SPRITES ZEIGEN ?
4330 :
4340 : BANK 15 : REM I/O-BANK AUSWAEHLEN
4350 :
4360 : Z=0 : REM SPRITE-ZEIGER AUF NULL
4370 : FOR I=0 TO 280 STEP 40 : REM ZEILEN
4380 :      FOR J=0 TO 7 : REM SPALTEN
4390 :          A1=PEEK(3648+Z) AND 2↑(7-J) : REM 1. WERT
4400 :          A2=PEEK(3712+Z) AND 2↑(7-J) : REM 2. WERT
4410 :          IF A1= 0 THEN Z1=224:K1=C1 : REM BIT EINS?
4420 :          IF A1<>0 THEN Z1=160:K1=C2 : REM BIT NULL?
4430 :          IF A2= 0 THEN Z2=224:K2=C1 : REM BIT EINS?
4440 :          IF A2<>0 THEN Z2=160:K2=C2 : REM BIT NULL?
4450 :          POKE  1533+I+J,Z2 : REM VERGROESSERN
4460 :          POKE 55805+I+J,K2 : REM AUCH FARBE SETZEN
4470 :          POKE  1520+I+J,Z1 : REM VERGROESSERN
4480 :          POKE 55792+I+J,K1 : REM UND FARBE SETZEN
4490 :      NEXT J
4500 :      Z=Z+3 : REM ZEIGER UM DREI ERHOEHEN
4510 : NEXT I
4520 :
4530 SLOW : REM SLOW-MODUS
4540 :
4550 GOTO 1120 : REM ZURUECK
4560 :
4570 REM **************************
4580 REM *   ZEICHEN UEBERNEHMEN  *
4590 REM **************************
```

```
4600 :
461Ø AD=8192+CD*8 : REM ADRESSE NEUES ZEICHEN
462Ø BANK Ø : REM RAM-BANK NULL AUSWAEHLEN
463Ø :
464Ø Z=Ø : REM SPRITEZEIGER AUF NULL
465Ø FOR I=Ø TO 7 : REM BYTE-ZAEHLER
466Ø :    POKE AD+I,PEEK (3776+Z) : REM UEBERNEHMEN
467Ø :    Z=Z+3 : REM SPRITEZEIGER AUF NAECHSTE ZEILE
468Ø NEXT I
469Ø :
47ØØ GOTO 419Ø : REM ZEICHEN DARSTELLEN
471Ø :
472Ø REM ***********************
473Ø REM *   FELDER KOPIEREN   *
474Ø REM ***********************
475Ø :
476Ø IF A$="2" THEN AD=152Ø : REM FELD ZWEI NACH EINS
477Ø IF A$="3" THEN AD=1533 : REM FELD DREI NACH EINS
478Ø :
479Ø FAST : REM FAST-MODUS
48ØØ : FOR I=Ø TO 28Ø STEP 4Ø : REM ZEILEN
481Ø :     FOR J=Ø TO 7 : REM SPALTEN
482Ø :          T=PEEK(AD+I+J) : REM WERT AUSLESEN
483Ø :          IF T<>16Ø THEN T=32 : REM SPACEFALG?
484Ø :          POKE 15Ø7+I+J,T : REM KOPIEREN
485Ø :     NEXT J
486Ø : NEXT I
487Ø SLOW : REM SLOWMODUS
488Ø :
489Ø GOTO 112Ø : REM ZUR TASTATURABFRAGE
49ØØ :
491Ø END
492Ø :
493Ø REM *****************************
494Ø REM *   DATA'S FUER ASSEMBLER   *
495Ø REM *         ROUTINEN          *
496Ø REM *****************************
497Ø :
498Ø DATA  76,  4, 11,  Ø,169,227
499Ø DATA 16Ø,  5,133,25Ø,132,251
5ØØØ DATA 169,192,16Ø, 14,133,252
5Ø1Ø DATA 132,253,162,  Ø,16Ø,  Ø
5Ø2Ø DATA 169,  Ø,141,  3, 11,177
5Ø3Ø DATA 25Ø,2Ø1,16Ø, 56,24Ø,  1
5Ø4Ø DATA  24,173,  3, 11, 42,141
5Ø5Ø DATA   3, 11,2ØØ,192,  8,2Ø8
5Ø6Ø DATA 236,173,  3, 11,16Ø,  Ø
5Ø7Ø DATA 145,252, 24,165,25Ø,1Ø5
5Ø8Ø DATA  4Ø,133,25Ø,169,  Ø,1Ø1
5Ø9Ø DATA 251,133,251, 24,165,252
51ØØ DATA 1Ø5,  3,133,252,169,  Ø
511Ø DATA 1Ø1,253,133,253,232,224
512Ø DATA   8,2Ø8,191, 96,  1
```

Programmbeschreibung:

100 - 130:	Programmkopf.
140 - 170:	Definierung der Farben und der Farbvariablen.
180 - 270:	Einlesen der Maschinenroutine mit Prüfsummentest.
280 - 450:	Sprite-Matrizen definieren.
460 - 540:	Kopieren des alten Zeichensatzes nach 8192 ($2000).
550 - 650:	Spriteparameter setzen.
660 - 740:	Positionieren der Sprites.
750 - 770:	Definierung des String EC$ mit dem Escape-code und Dimensionierung eines Arrays für Rotieren und Scrollen.
780 - 1100:	Bildschirmaufbau mit anschließendem Sprung zur Farbausgabe.
1110 - 1140:	Definition des Eingabefensters als Window und Initialisieren der Cursor-Koordinaten.
1150 - 1180:	Unterprogrammkopf.
1190 - 1210:	Cursor bewegen und Eingabefeld als Sprite darstellen.
1220 - 1790:	Tastaturabfrage.
1800 - 1940:	Eingabefeld invertieren.
1950 - 2110:	Eingabefeld an der vertikalen Achse spiegeln.
2120 - 2280:	Eingabefeld an der horizontalen Achse spiegeln.
2290 - 2540:	Zeichenmatrix im Uhrzeigersinn rotieren.
2550 - 2660:	Abfragen, in welche Richtung gerollt werden soll.
2670 - 2890:	Linksscrollen.
2900 - 3080:	Rechtsscrollen.
3090 - 3250:	Obenscrollen.
3260 - 3420:	Untenscrollen.
3430 - 3450:	Rücksprung in übergeordnete Routine.
3460 - 3560:	Zeichensatz laden.
3570 - 3670:	Zeichensatz speichern.
3680 - 4030:	Farben anzeigen und Spritefarben anpassen.
4040 - 4170:	Codenummer zentrieren und ausgeben.
4180 - 4550:	Feld 2 und 3 vergrößert darstellen.
4560 - 4700:	Zeichen in neuen Zeichensatz übernehmen.
4710 - 4910:	Feld 2 oder 3 ins Eingabefeld kopieren.
4920 - 5120:	Data's für Assemblerunterroutinen.

2.4.3.7 Verschieben des Textschirms

Eigentlich müßte man zwei Kapitel über die Verschiebung des Textschirms schreiben, denn es besteht ein sehr großer Unterschied zwischen der reinen physikalischen Lage des Video-RAMs und der softwaremäßigen Verschiebung des Textschirms. Hardwaremäßig kann man in beiden Modi, dem 64er und dem 128er Modus den Textschirm an fast jede Adresse innerhalb des VIC-Adreßraums legen, doch softwaremäßig bei weitem nicht an jede. Im 64er Modus ist es harware- als auch softwaremäßig möglich, den Textschirm an fast jede beliebige Adresse zu legen, vorausgesetzt, das Betriebssystem kann diese lesen und beschreiben. Für die softwaremäßige Umschaltung im 64er Modus ist nämlich nur ein einziges Register, nämlich das Register 648 zuständig. Es enthält das Highbyte der Adresse, an der das Video-RAM beginnt und errechnet sich nach folgender Formel:

Highbyte = INT (Adresse / 256)

Haben Sie das Video-RAM also physikalisch nach 2048 ($0800) verlegt, so müssen Sie den Wert 8 ($08) in die Adresse 648 schreiben, um auch softwaremäßig auf diesen Bildschirm zuzugreifen.

Im Gegensatz zum 64er Modus ist es unseres Wissens nach softwaremäßig leider nicht möglich, den Textschirm im 128er-Modus zu verschieben, da die Low- und auch die Highbytes der Bildschirmzeilenanfänge im ROM liegen und daher nicht verändert werden können.

Als ob Commodore wieder etwas hätte gut machen wollen, läßt sich die hardwaremäßige Verschiebung in beiden Modi wunderbar bewerkstelligen. Im 128er Modus können wir sogar noch festlegen, aus welcher Speicherbank der VIC seine Informationen beziehen soll (siehe auch Kapitel 2.2). Doch nun zum prinzipiellen Umschalten des Video-RAM -Adresse, wie es

auch bei der HI-RES Grafik benutzt wird. Zur Umschaltung wird, wie könnte es auch anders sein, wieder einmal das Register 24 des Videocontrollers verwendet. Mit den obersten vier Bits (4-7) werden die Adreßbits 10 bis 13 kodiert, das Video-RAM kann also maximal in 2^13 gleich 8192 ($2000) Byteschritten verschoben werden. Minimal beträgt die Auflösung der Verschiebung 2^10 gleich 1024 ($0400) Bytes. Beispiel: Wollen Sie den Textschirm nach 9216 ($2400) legen, so müßten Sie die folgenden der oberen vier Bits im Register 24 setzen:

$$9216 = 2^{10} + 2^{13}$$

Also die Bits 4 und 7. Soviel nun zur Art und Weise wie der Textschirm verschoben wird. Zur Demonstration der phänomenalen Vorteile, die sich hiermit ergeben, hier ein kleiner Vorgeschmack. Das nun folgende Assemblerprogramm (nur 64er) verwaltet drei Textschirme, die bei 1024 ($0400),2048 ($0800) und 3072 ($0C00) liegen und mittels der Funktionstasten F1,F3 und F5 angesprochen werden. Der Basicspeicher wird bei der Initialisierung des Programms nach oben verschoben, Sie sollten daher das Programm auf jeden Fall vor dem Starten abspeichern, da der Basicspeicher gelöscht wird.

Assemblerlisting:

```
profi-ass 64 v2.0

110:   c000                       .opt p1,oo
120:                    ;
130:                    ;*********************************
140:                    ;*   programm zur verwaltung    *
150:                    ;*   von drei textschirmen      *
160:                    ;*********************************
170:                    ;
180:   c000                       *=   $c000
190:                    ;
194:   c000 4c 05 c0              jmp  start
195:                    ;
200:   d000             v         =    53248   ; basisadresse vic ii
```

```
210:   d800           farbe    =    55296    ; adresse farbram
215:   c003 00        altfunkt .byte0        ; letzte taste
216:   c004 00        mem      .byte0        ; letzte taste
220:                  ;
230:   c005 78        start    sei           ;interrupt verhindern
240:   c006 a9 3b              lda  #<neuirq ;low-byte
250:   c008 a0 c0              ldy  #>neuirq ;high-byte
260:   c00a 8d 14 03           sta  $0314    ;im sprungvektor fuer
270:   c00d 8c 15 03           sty  $0315    ;irq speichern
280:   c010 58                 cli           ;interrupt freigeben
281:   c011 a9 ff              lda  #$ff     ;startwert
282:   c013 8d 03 c0           sta  altfunkt ;in zwischenspeicher
283:   c016 a9 01              lda  #<$2001 ;neuer programmstart low
284:   c018 a0 20              ldy  #>$2001 ;neuer programmstart high
285:   c01a 85 2b              sta  43       ;als neuen basicstart setzen
286:   c01c 84 2c              sty  44
287:   c01e 85 f7              sta  $f7      ;auch in $f7/$f8
289:   c020 84 f8              sty  $f8      ;schreiben
290:   c022 38                 sec           ;carry fuer sbc setzen
291:   c023 e9 01              sbc  #1       ;minus 1
292:   c025 85 f7              sta  $f7      ;in $f7
293:   c027 b0 02              bcs  ok1      ;wenn kein unterlauf dann
nach ok1
294:   c029 c6 f8              dec  $f8      ;high-byte erniedrigen
295:   c02b a0 00     ok1      ldy  #0       ;index auf null
296:   c02d a9 00              lda  #0       ;akku mit null laden
297:   c02f 91 f7              sta  ($f7),y ;and basicstart schreiben
298:   c031 60                 rts           ;zurueck zum basic
300:                  ;
310:   c032 04 08 0c tab1      .byte$04,$08,$0c ;tabelle bildschirmseiten
320:   c035 15 25 35 tab2      .byte21, 37, 53 ;tabelle werte reg.24
330:   c038 10 14 18 tab3      .byte$10,$14,$18 ;tabelle farbramseiten
340:                  ;
350:   c03b a5 c5     neuirq   lda  197      ;tastaturcode holen
360:   c03d 38                 sec           ;carry fuer sbc setzen
370:   c03e e9 04              sbc  #4       ;vier subtrahieren
380:   c040 c9 03              cmp  #3       ;groesser als drei
390:   c042 b0 53              bcs  nofunkt ;ja, dann ende
395:   c044 cd 03 c0           cmp  altfunkt ;gleich letzter
funktionstaste
```

```
396:   c047 f0 4e              beq  nofunkt ;ja, dann ende
397:   c049 8d 04 c0           sta  mem     ;nein, dann zwischenspeichern
400:   c04c a8                 tay          ;akku als zeiger ins yr
410:   c04d b9 32 c0           lda  tab1,y  ;bildschirmseite laden
420:   c050 8d 88 02           sta  648     ;und als msb setzen
430:   c053 b9 35 c0           lda  tab2,y  ;wert fuer reg.24 laden
440:   c056 8d 18 d0           sta  v+24    ;und in reg.24 schreiben
450:                     ;
460:   c059 b9 38 c0            lda  tab3,y  ;farbramseite (kopie) = high-
byte
470:   c05c a0 00              ldy  #0      ;low-byte = 0
480:   c05e 84 f7              sty  $f7     ;als zeiger-low
490:   c060 85 f8              sta  $f8     ;high
500:                     ;
510:   c062 a9 00              lda  #<farbe ;adresse low farbram
520:   c064 a0 d8              ldy  #>farbe ;high
530:   c066 85 f9              sta  $f9     ;als zeiger
540:   c068 84 fa              sty  $fa     ;setzen
550:                     ;
560:   c06a ac 03 c0           ldy  altfunkt ;letzte funktionstaste
570:   c06d b9 38 c0           lda  tab3,y  ;msb farbramkopie
580:   c070 a0 00              ldy  #0      ;low-byte = 0
590:   c072 84 fb              sty  $fb     ;als zeiger
600:   c074 85 fc              sta  $fc     ;speichern
610:                     ;
620:   c076 a2 04              ldx  #4      ;4 mal 256
630:   c078 a0 00     loop0    ldy  #0      ;yr gleich 0
640:   c07a b1 f9     loop1    lda  ($f9),y ;aus farbram
650:     c07c 91  fb                 sta   ($fb),y ;in kopie von letzte
funkt.taste
660:   c07e b1 f7              lda  ($f7),y ;neue kopie
670:   c080 91 f9              sta  ($f9),y ;ins farbram
680:   c082 c8                 iny          ;yr-erhoehen
690:   c083 d0 f5              bne  loop1   ;ungleich null, dann sprung
700:   c085 e6 fa              inc  $fa     ;highbytes
710:   c087 e6 fc              inc  $fc     ;erhoehen
720:   c089 e6 f8              inc  $f8
730:   c08b ca                 dex          ;seitenzaehler minus eins
740:   c08c d0 ea              bne  loop0   ;testen, ob fertig
```

```
745:    c08e 20 66 e5          jsr  $e566    ;ja, dann cursor-home
746:    c091 ad 04 c0          lda  mem      ;zwischenspeicher
747:    c094 8d 03 c0          sta  altfunkt ;als letzte funkt.taste
750:    c097 4c 31 ea nofunkt  jmp  $ea31    ;zum normalen irq
```

Basiclader:

```
100 rem ********************************
110 rem *   extended-screen-program    *
120 rem ********************************
130 :
140 for i=49152 to 49305
150 :   read a
160 :   s=s+a
170 :   poke i,a
180 next i
190 :
200 if s<>20299 then print "fehler in data's!!!": end
210 print "programm ok!!"
220 :
230 end : rem speichern !!!
240 :
10000 data  76,  5,192,  0,  0,120,169
10010 data  59,160,192,141, 20,  3,140
10020 data  21,  3, 88,169,255,141,  3
10030 data 192,169,  1,160, 32,133, 43
10040 data 132, 44,133,247,132,248, 56
10050 data 233,  1,133,247,176,  2,198
10060 data 248,160,  0,169,  0,145,247
10070 data  96,  4,  8, 12, 21, 37, 53
10080 data  16, 20, 24,165,197, 56,233
10090 data   4,201,  3,176, 83,205,  3
10100 data 192,240, 78,141,  4,192,168
10110 data 185, 50,192,141,136,  2,185
10120 data  53,192,141, 24,208,185, 56
10130 data 192,160,  0,132,247,133,248
10140 data 169,  0,160,216,133,249,132
10150 data 250,172,  3,192,185, 56,192
10160 data 160,  0,132,251,133,252,162
```

```
10170 data   4,160,  0,177,249,145,251
10180 data 177,247,145,249,200,208,245
10190 data 230,250,230,252,230,248,202
10200 data 208,234, 32,102,229,173,  4
10210 data 192,141,  3,192, 76, 49,234
```

2.5 IMR/IRR-Interruptprogrammierung

Wir werden Ihnen jetzt erklären, was man alles mit diesen beiden sagenumwobenen VIC-Registern machen kann. Die Chancen stehen 100 zu 1, daß auch Sie völlig fasziniert sein werden von den Perspektiven, die sich Ihnen eröffnen. Doch dazu gehört auch trockene Theorie, die wir jetzt möglich schnell durchziehen wollen:

Die beiden Begriffe IRR und IMR haben, wie Sie vielleicht schon der Registerbeschreibung entnehmen konnten, beide etwas mit der Interruptauslösung durch den Videocontroller zu tun. Zu Deutsch bedeuten die beiden Registerabkürzungen in etwa:

IRR: Nachfrageregister für die Art des vom VIC ausgelösten Interrupts (IRQ).

IMR: Festlegeregister für die VIC-Funktionen, die einen IRQ aulösen dürfen.

Doch für die, die nicht genau wissen was ein Interrupt ist, hier erst einmal eine Erklärung dieses Begriffes:

Unter einem Interrupt versteht man die gesteuerte Programmunterbrechung an einer beliebigen Stelle, verursacht durch irgendein Ereignis. Ist eine solche Unterbrechung ausgelöst worden, so springt der Computer sofort über einen Vektor in die sogenannte Interruptroutine. Nach Beendigung dieser Interruptroutine springt er zum eigentlichen Programm zurück und fährt

dort in gewohnter Weise fort. Ein Interrupt ist also eine gewollte und programmtechnisch vorgesehene Unterbrechung des Programms, wobei nach der Abarbeitung der Interruptroutine wieder an der alten Programmstelle fortgefahren wird. Ihr Commodore 128 kennt folgende vier Interruptmöglichkeiten:

- NMI (Non Maskable Interrupt: nicht programmtechnisch auszuschaltender Interrupt)

- BRK (Break)

- IRQ (Interrupt Request)

- Reset

a) NMI
Ein NMI wird ausgelöst durch das Drücken der RESTORE-Taste oder beim Betreiben einer RS-232C Schnittstelle. Er kann softwaremäßig nicht unterdrückt werden, wird also in jedem Fall ausgelöst. Falls Sie eine RS-232C Schnittstelle besitzen, oder die RESTORE-Taste umfunktionieren wollen, so können Sie den Vektor auf die NMI-Routine in den Vektoren $0318/$0319 ändern.

b) BRK
Dieser Interrupt wird nicht hardwaremäßig gesteuert, sondern wird durch einen bestimmten Assemblerbefehl (BRK) ausgelöst. Der Prozessor fährt dann mit seiner Arbeit in der IRQ-Routine fort, in der getestet wird, ob es sich um einen IRQ oder um einen BRK handelt.

c) IRQ
Der IRQ stellt in etwa die hardwaremäßige Parallele zum BRK Interrupt dar, nur daß der IRQ makierbar ist, im Gegensatz zum BRK. Wird er ausgelöst (z.B. durch die CIA1 oder den VIC), so wird indirekt über die Vektoren $0314/$0315 in die Interruptroutine gesprungen.

d) Reset
Der Reset Interrupt kann nicht softwaremäßig unterdrückt werden und wird nach dem Einschalten des Computers oder nach dem Betätigen der Reset-Taste ausgelöst.

Für uns ist in Bezug auf das IMR und das IRR nur der durch den VIC ausgelöste IRQ interessant. Der VIC-Interrupt läßt sich noch in vier Interruptursachen aufteilen, die da lauten:

- Rasterzeile
- Lightpen
- Sprite-Spritekollision
- Sprite-Hintergrundkollision

Wie weiter oben zu lesen war, springt der Prozessor bei einem IRQ indirekt über die Vektoren $0314/$0315 in die IRQ-Routine. Genau an dieser Stelle können wir nun eingreifen. Wir können z.B. eine Routine einbauen, die Überprüft, ob es sich um einen vom Videocontroller ausgelösten Interrupt oder um einen Systeminterrupt handelt. Im ersten Fall könnten wir dann weiter zwischen den vier verschiedenen Möglichkeiten unterscheiden und entsprechend reagieren. Im zweiten Fall müßten wir einfach wieder zur normalen IRQ-Routine springen, wobei wir aber das Interruptflag wieder freigeben müssen, da ja auch innerhalb der IRQ-Routine ein Interrupt auftreten kann, der sonst vernachlässigt würde.

Nun zur Programmierung des IMR und des IRR:
Im IMR-Register können Sie festlegen, bei welchem Ereignis der VIC einen Interrupt auslösen soll. Die Aufschlüsselung dieses Registers entnehmen Sie bitte der Registerbeschreibung. Wollen wir ein Bit im IMR setzen, dieses Ereignis also als Interruptursache zulassen, so schreiben wir einfach dieses Bit mit gesetztem 7. Bit in das IMR. Beispiel: Wollen wir als Interruptursache den Rasterinterrupt zulassen, so lautete der Wert, den wir in das IMR schreiben müßten %10000001 also 129. Das Löschen der einzelnen Bits muß analog erfolgen, wir schreiben dann nur die zu löschenden Bits ohne das siebte Bit in das IMR. Beispiel:

Wollten wir den Rasterinterrupt wieder löschen, so müßten wir folgenden Wert in das Register schreiben: %00000001 also 1.

Nun zum Aufbau des IRR:
Die Kodierung der einzelnen Bits des IRR ist identisch mit dem Aufbau des IMR. Nur Bit sieben besitzt eine andere Funktion. Dieses ist gesetzt, wenn eines der anderen Bits gesetzt ist, also ein Interrupt durch den Videocontroller ausgelöst wurde. Ein Löschen dieses Registers, das nach jedem VIC-Interrupt erfolgen muß, ist einfach durch das Zurückschreiben des gelesenen Wertes möglich. Nun zur Anwendung des IRR bzw. IMR. Damit Sie einen Eindruck erhalten, wollen wir hier erst einmal einen kleinen Bereich der Dinge aufzählen, die sich mit dem VIC-Interrupt programmieren lassen:

- Kollisionsabfrage
- Sprite-Hintergrundzusammenstoß
- Mehrere VIC-Simulationen- Text/Grafik/MC gemischt
- 40 Sprite (in Worten: Vierzig)
- Mehrere Zeichensätze
- 25% Geschwindigkeitserhöhung

usw.

Einige dieser Verwendungsmöglichkeiten kennen Sie schon in geringen Ausmaßen vom BASIC 7.0. So geschieht z. B. die Kollisionsabfrage oder das Splitting des Bildschirms mit dem VIC-Interrupt.
Doch damit sind noch lange nicht alle Möglichkeiten des VIC-Interrupts ausgeschöpft. Wir wollen an dieser Stelle nur noch genauer auf den Rasterinterrupt eingehen, da dieser praktisch der Schlüssel zu allen Effekten ist.

Um einen Interrupt durch einen Rasterzeilendurchlauf auszulösen, muß erst einmal geklärt werden, was unter einer Rasterzeile zu verstehen ist. Wie Sie vielleicht wissen, wird ein Monitorbild mittels eines Elektronenstrahls aufgebaut, der VIC muß das Bild also auch punktweise zur Verfügung stellen. Dabei wird der Bildschirm in eine bestimmte Anzahl an Zeilen aufgeteilt, und zwar vom oberen Bildschirmrand bis zum unteren (nicht Textfenster!!!). Diese Zeilen bezeichnet man als sogenannte Raster-

zeilen, da sie das Raster für die Y-Auflösung darstellen. Der VIC stellt nun ein Register zur Verfügung, in dem die Nummer der momentan aufgebauten Rasterzeile enthalten ist. Und noch besser: Der Programmierer kann sogar festlegen, bei welcher Rasterzeile ein IRQ ausgelöst werden soll!!!
Mit diesem Verfahren ließe sich z. B. bewerkstelligen, daß das Bild bis zur Rasterzeile 100 im Einzelpunktmodus aufgebaut wird, dann ein IRQ ausgelöst wird und in der IRQ-Routine dann auf den Textmodus umgeschaltet wird. Dieses Prinzip wird auch bei der Befehlsanweisung GRAPHIC 2,X,Z benutzt. Nun stellen Sie sich doch einmal vor, daß man bei einem bestimmten Rasterzeilendurchlauf nicht nur ein Register, sondern sogleich den ganzen VIC umregistrieren würde. Welche Effekte sich damit erreichen ließen, brauchen wir Ihnen wohl gar nicht weiter erklären.

2.5.1 5 * VIC II

Nachdem wir nun schon im letzten Kapitel geklärt haben, wie der VIC-Interrupt programmiert wird, wollen wir Ihnen hier eine universelle Lösung zur Programmierung des Rasterinterrupts vorstellen. Es handelt sich dabei um ein komplettes VIC-Simulationsprogramm, bei dem an bestimmten Rasterzeilen nicht nur einzelne Werte, sondern sogleich der ganze VIC umregistriert wird. Diese Umregistrierung ist an fünf frei definierbaren Rasterzeilen möglich. Doch erst einmal zur Bedienung des Programms. Nachdem Sie den Lader gestartet haben, werden die DATA's eingelesen, das Programm aber noch nicht gestartet. Dies geschieht erst beim ersten Festlegen einer Rasterzeile, bei deren Strahlendurchlauf auf eine andere Kopie zugegriffen werden soll. Das Festlegen der Rasterzeilen geschieht auf folgende Art:

SYS 49152,Kopie,Rasterzeile

Ihnen stehen hierbei die Kopien 1 bis 5 zur Verfügung. Jeder andere Wert außer Null, der zum Ausschalten des Extended-VIC

Programms dient, führt zu einem Fehler. Die Rasterzeilen sind im Bereich von 0 bis 511 zu wählen, wobei darauf geachtet werden sollte, daß die jeweils höher gelegene VIC-Kopie auch bei der höheren Rasterzeile eingelesen wird, also beispielsweise so:

SYS 49152,1,50
SYS 49152,2,100
SYS 49152,3,150

usw.

Dagegen führt folgendes zu einem nervtötenden Bildschirmflimmern:

SYS 49152,1,150
SYS 49152,2,50

Achten Sie also auf die Werte, die Sie wählen. Vor allen I/O Operationen sollten Sie das Programm durch RUN/STOP RESTORE oder SYS 49152,0 desaktiviern, da es sonst zu Timingproblemen kommen kann. Beeinflussen Sie auch das VIC-Register 48 im Rasterirq (warum dazu später), müssen Sie beachten, daß das 64er Betriebssystem dieses Register beim Drücken der RESTORE-Taste nicht zurücksetzt, Sie dies also folgendermaßen von Hand erledigen müssen:

POKE 53248+48,0 (RETURN)

Hier nun zuerst das dokumentierte Assemblerlisting und dann der Basiclader, der im 64er Modus eingegeben werden muß:

Assemblerlisting:

```
profi-ass 64 v2.0

110:    c000                    .opt p1,oo
120:                 ;
```

```
130:                    ;******************************
140:                    ;*   extended-vic programm    *
150:                    ;******************************
160:                    ;
170:  c000                      *=   $c000
180:                    ;
190:  c000 4c 06 c0             jmp  start    ; zum programmstart
200:                    ;
210:  b79e              getbyt  =    $b79e    ; byte holen
220:  ad8a              frmnum  =    $ad8a    ; formel auswerten
230:  b7f7              getadr  =    $b7f7    ; 16-bit wert holen
240:  aefd              chkcom  =    $aefd    ; auf komma testen
250:  0079              chrgot  =    $0079    ; letztes zeichen
260:  0073              chrget  =    $0073    ; naechstes zeichen
270:                    ;
280:  d000              v       =    53248    ; basisadresse vic
290:                    ;
300:  c003 4c 48 b2 error       jmp  $b248    ; illegal quantity
310:                    ;
320:  c006 20 fd ae start       jsr  chkcom   ; komma holen
330:  c009 20 9e b7             jsr  getbyt   ; byte wert holen
340:  c00c e0 06                cpx  #6       ; testen ob erlaubt
350:  c00e b0 f3                bcs  error    ; zahl zu hoch
360:  c010 e0 00                cpx  #0       ; null
370:  c012 f0 11                beq  off      ; bei null irq aus
380:  c014 8a                   txa
380:  c015 48                   pha
380:  c016 ad 14 03             lda  $0314
390:  c019 c9 b5                cmp  #<neuirq  ; schon raster aktiv
400:  c01b f0 03                beq  ok       ; ja
410:  c01d 20 37 c0             jsr  init     ; raster initialisieren
420:  c020 68           ok      pla
420:  c021 aa                   tax
420:  c022 4c 80 c0             jmp  setirq
430:                    ;
440:                    ;***********
450:                    ;*   off   *
460:                    ;***********
470:                    ;
480:  c025 78           off     sei           ; interrupt verhindern
```

```
490:   c026 a9 31          lda  #<$ea31  ; low-byte irq
500:   c028 a0 ea          ldy  #>$ea31  ; high-byte irq
510:   c02a 8d 14 03       sta  $0314   ; interruptvektor low
520:   c02d 8c 15 03       sty  $0315   ; interruptvektor high
530:   c030 a9 80          lda  #%10000000  ; maske fuer imr
540:   c032 8d 1a d0       sta  v+26    ; und loeschen
550:   c035 58             cli          ; interrupt freigeben
560:   c036 60             rts          ; wieder zum basic
570:                 ;
580:                 ;**********************************
590:                 ;* initialisieren des raster-irq *
600:                 ;**********************************
610:                 ;
620:   c037 78       init  sei          ; interrupt verhindern
630:   c038 a9 b5          lda  #<neuirq  ; low-byte neuirq
640:   c03a a0 c0          ldy  #>neuirq  ; high byte neuirq
650:   c03c 8d 14 03       sta  $0314   ; in irq-vektor low
660:   c03f 8c 15 03       sty  $0315   ; in irq-vektor high
670:   c042 a0 00          ldy  #0      ; yr-index auf null
680:   c044 b9 00 d0 loop0 lda  v,y     ; orginalen vic
690:   c047 c0 11          cpy  #17     ; register 17
700:   c049 d0 04          bne  ok1     ; nein!
710:   c04b 29 7f          and  #%01111111  ; bit 7 loeschen
720:   c04d d0 0c          bne  ok9     ; nach ok2
730:   c04f c0 12    ok1   cpy  #18     ; register 18
740:   c051 d0 02          bne  ok2     ; nein!
750:   c053 a9 00          lda  #0
760:   c055 c0 1a    ok2   cpy  #26
760:   c057 d0 02          bne  ok9
760:   c059 a9 81          lda  #%10000001
770:   c05b 99 f7 c0 ok9   sta  vic1,y
780:   c05e 99 28 c1       sta  vic2,y  ; in kopie iii
790:   c061 99 59 c1       sta  vic3,y  ; in kopie ii
800:   c064 99 8a c1       sta  vic4,y  ; in kopie  iv
810:   c067 99 bb c1       sta  vic5,y  ; in kopie   v
820:   c06a c8             iny          ; index erhoehen
830:   c06b c0 31          cpy  #49     ; schon alle register
840:   c06d d0 d5          bne  loop0   ; nein!
850:                 ;
860:   c06f a9 00          lda  #0      ; vic kopie 0
```

```
870:   c071 85 02                sta  2          ; index-zeiger
880:   c073 a9 81                lda  #%10000001  ; raster irq
890:   c075 8d 1a d0             sta  v+26       ; ins imr schreiben
900:   c078 ad 19 d0             lda  v+25       ; irr auslesen
910:   c07b 8d 19 d0             sta  v+25       ; und loeschen
920:   c07e 58                   cli             ; interrupt freigeben
930:   c07f 60                   rts             ; und wieder zurueck
940:                   ;
950:                   ;**********************************
960:                   ;*   zeile fuer irq festlegen    *
970:                   ;**********************************
980:                   ;
990:   c080 8a         setirq    txa             ; und nach akku bringen
1000:  c081 0a                   asl             ; verdoppeln und
1010:  c082 aa                   tax             ; wieder nach x bringen
1020:  c083 bd eb c0             lda  tab1,x     ; low-byte vic-kopie
1030:  c086 85 f7                sta  $f7
1040:  c088 bd ec c0             lda  tab1+1,x   ; high-byte vic-kopie
1050:  c08b 85 f8                sta  $f8
1060:                  ;
1070:  c08d 20 fd ae             jsr  chkcom     ; auf komma testen
1080:  c090 20 8a ad             jsr  frmnum     ; formelauswertung
1090:  c093 20 f7 b7             jsr  getadr     ; 16-bit wert holen
1100:  c096 a5 15                lda  $15        ; highbyte 16-bitwert
1110:  c098 c9 02                cmp  #2         ; groesser gleich 2
1120:  c09a 90 03                bcc  ok4        ; in ordnung
1130:  c09c 4c 03 c0             jmp  error      ; illegal quantity
1140:  c09f 78         ok4       sei             ; interrupt sperren
1150:  c0a0 4a                   lsr  a          ; bit 0 ins carry
1160:  c0a1 6a                   ror  a          ; carry ins 7. bit
1170:  c0a2 85 f9                sta  $f9        ; in adr. $f9 speichern
1180:  c0a4 a0 11                ldy  #17        ; register 17
1190:  c0a6 b1 f7                lda  ($f7),y    ; register 17 laden
1200:  c0a8 29 7f                and  #%01111111
1210:  c0aa 05 f9                ora  $f9        ; oder verknuepfen
1220:  c0ac 91 f7                sta  ($f7),y
1230:  c0ae a5 14                lda  $14        ; low-byte
1240:  c0b0 c8                   iny             ; register 18
1250:  c0b1 91 f7                sta  ($f7),y
1260:  c0b3 58                   cli             ; interrupt freigeben
```

```
1270:  c0b4 60                 rts           ; wieder zum basic
1280:                ;
1290:                ;************************
1300:                ;*   neue irq-routine   *
1310:                ;************************
1320:                ;
1330:  c0b5 ad 19 d0 neuirq    lda  v+25
1330:  c0b8 8d 19 d0           sta  v+25
1340:  c0bb 30 07              bmi  raster
1340:  c0bd ad 0d dc           lda  $dc0d
1340:  c0c0 58                 cli
1340:  c0c1 4c 31 ea           jmp  $ea31
1350:  c0c4 e6 02    raster    inc  2        ; index erhoehen
1360:  c0c6 a5 02              lda  2        ; einladen
1370:  c0c8 c9 05              cmp  #5       ; schon sechs
1380:  c0ca d0 04              bne  ok3      ; nein!
1390:  c0cc a9 00              lda  #0       ; index auf null setzen
1400:  c0ce 85 02              sta  2        ; und abspeichern
1410:  c0d0 0a       ok3       asl           ; verdoppeln
1420:  c0d1 a8                 tay           ; akku nach yr
1430:  c0d2 b9 eb c0           lda  tab1,y   ; low-byte kopie
1440:  c0d5 85 fa              sta  $fa
1450:  c0d7 b9 ec c0           lda  tab1+1,y
1460:  c0da 85 fb              sta  $fb
1470:  c0dc a0 00              ldy  #0       ; index loeschen
1480:  c0de b1 fa    loop3     lda  ($fa),y  ; kopie in
1490:  c0e0 99 00 d0           sta  v,y      ; vic kopieren
1500:  c0e3 c8                 iny           ; index erhoehen
1510:  c0e4 c0 31              cpy  #49      ; alle register
1520:  c0e6 d0 f6              bne  loop3    ; nein
1530:  c0e8 4c bc fe           jmp  $febc
1540:                ;
1550:  c0eb bb c1 f7 tab1      .wordvic5,vic1,vic2,vic3,vic4,vic5
1560:                ;
1570:  c0f7          vic1      =    *
1580:  c128          vic2      =    vic1+49
1590:  c159          vic3      =    vic2+49
1600:  c18a          vic4      =    vic3+49
1610:  c1bb          vic5      =    vic4+49
1620:                ;
```

Basiclader:

```
100 rem ******************************
110 rem *   extended-vic-program    *
120 rem ******************************
130 :
140 for i=49152 to 49398
150 :   read a
160 :   s=s+a
170 :   poke i,a
180 next i
190 :
200 if s<>32344 then print "fehler in data's!!!" : end
210 :
220 print "programm ok!!!"
230 :
240 end
250 :
260 data  76,  6,192, 76, 72,178, 32
270 data 253,174, 32,158,183,224,  6
280 data 176,243,224,  0,240, 17,138
290 data  72,173, 20,  3,201,181,240
300 data   3, 32, 55,192,104,170, 76
310 data 128,192,120,169, 49,160,234
320 data 141, 20,  3,140, 21,  3,169
330 data 128,141, 26,208, 88, 96,120
340 data 169,181,160,192,141, 20,  3
350 data 140, 21,  3,160,  0,185,  0
360 data 208,192, 17,208,  4, 41,127
370 data 208, 12,192, 18,208,  2,169
380 data   0,192, 26,208,  2,169,129
390 data 153,247,192,153, 40,193,153
400 data  89,193,153,138,193,153,187
410 data 193,200,192, 49,208,213,169
420 data   0,133,  2,169,129,141, 26
430 data 208,173, 25,208,141, 25,208
440 data  88, 96,138, 10,170,189,235
450 data 192,133,247,189,236,192,133
```

```
460 data 248, 32,253,174, 32,138,173
470 data  32,247,183,165, 21,201,  2
480 data 144,  3, 76,  3,192,120, 74
490 data 106,133,249,160, 17,177,247
500 data  41,127,  5,249,145,247,165
510 data  20,200,145,247, 88, 96,173
520 data  25,208,141, 25,208, 48,  7
530 data 173, 13,220, 88, 76, 49,234
540 data 230,  2,165,  2,201,  5,208
550 data   4,169,  0,133,  2, 10,168
560 data 185,235,192,133,250,185,236
570 data 192,133,251,160,  0,177,250
580 data 153,  0,208,200,192, 49,208
590 data 246, 76,188,254,187,193,247
600 data 192, 40,193, 89,193,138,193
610 data 187,193
```

Nachdem wir nun schon ein sehr komfortables Programm zum Arbeiten mit dem VIC-Interrupt haben, wollen wir dieses Programm auch entsprechend nutzen. Splitten Sie den Bildschirm bitte mit folgenden fünf Kommandos in fünf etwa gleich große Rasterzonen:

SYS 49152,1,40
SYS 49152,2,90
SYS 49152,3,140
SYS 49152,4,190
SYS 49152,5,240

Dann definieren Sie bitte die Anfangsadressen der einzelnen VIC-Kopien wie folgt (Direktmodus):

V1=49399:V2=V1+49:V3=V2+49:V4=V3+49:V5=V4+49

Nun können Sie die einzelnen "Pseudovics" wie gewohnt verändern, nur daß Sie jeweils eine andere Basisadresse benutzen müssen. Für die erste VIC-Kopie V1, für die zweite Kopie V2 usw.

Teilen wir unseren Bildschirm doch einmal in eine Text, eine HGR und eine MC Screen. Dies könnte beispielsweise so aussehen (Basisadresse in den Variablen V1-V5):

POKE V2+17,PEEK (V1+17) OR 2^5 : rem grafik aktivieren
POKE V2+24,24 : rem grafikspeicher bei 8192 ($2000)
POKE V3+17,PEEK (V2+17) OR 2^5 : rem grafik aktivieren
POKE V3+24,24 : rem grafikspeicher bei 8192 ($2000)
POKE V3+22,PEEK (V2+22) OR 2^4 : rem Multi-Color-Modus

Und schon stellen die oberen drei Bereiche HI-RES und die unteren beiden Bereiche Text dar. Um Ihnen zu demonstrieren, wie die Verwaltung von 32 Sprites aussehen kann, hier noch eine Spritedemo, die natürlich auch nur im 64er Modus läuft:

Spritedemo:

```
100 rem ******************************
110 rem *   extended-vic-program     *
120 rem *   --------demo-------      *
130 rem ******************************
140 :
150 print chr$(147):
160 :
170 v(1) =   49399 : rem adr. kopie (1)
180 v(2) = v(1)+49 : rem adr. kopie (2)
190 v(3) = v(2)+49 : rem adr. kopie (3)
200 v(4) = v(3)+49 : rem adr. kopie (4)
210 v(5) = v(4)+49 : rem adr. kopie (5)
220 :
230 sys 49152,1, 41 : rem irq-zeile (1)
240 sys 49152,2, 91 : rem irq-zeile (2)
250 sys 49152,3,141 : rem irq-zeile (3)
260 sys 49152,4,191 : rem irq-zeile (4)
270 sys 49152,5,240 : rem irq-zeile (5)
280 :
290 for i=1 to 5
300 :   for j=0 to 48
310 :       read a
```

```
320 :        if j=17 then a=a and 127
330 :        if j=18 then 350
340 :        poke v(i)+j,a
350 :    next j
360 next i
370 :
380 for i=0 to 62
390 :    read a
400 :    poke 704+i,a
410 next i
420 :
430 for i=2040 to 2047 : poke i,11 : next i
440 t=peek(v(5)+33)
450 for i=4 to 1 step -1
460 :    poke v(i+1)+33,peek(v(i)+33)
470 :    poke v(i+1)+32,peek(v(i)+32)
480 next i
490 :
500 poke v(1)+33,t
510 poke v(1)+32,t
520 :
530 for pa=0 to 250 : next pa
540 :
550 goto 440
560 :
570 rem ******************
580 rem *   vic data's   *
590 rem ******************
600 :
610 data   0,  0,  0,  0
620 data   0,  0,  0,  0
630 data   0,  0,  0,  0
640 data   0,  0,  0,  0
650 data   0, 27, 41,209
660 data   0,255,200,  0
670 data  21,121,129,255
680 data 255,  0,  0,255
690 data   0,  0,241,242
700 data 243,  2,  7,  6
710 data   6,  6,  6,  6
```

```
720 data   6,  6,  6,255
730 data 253
740 rem ---------------
750 data  50, 60, 79, 60
760 data 108, 60,137, 60
770 data 167, 60,196, 60
780 data 225, 60,254, 60
790 data   0, 27, 91,209
800 data   0,255,200,  0
810 data  21,121,129,255
820 data 255,  0,  0,255
830 data   2,  2,241,242
840 data 243,  2,  7,  6
850 data   6,  6,  6,  6
860 data   6,  6,  6,255
870 data 252
880 rem ---------------
890 data  50,110, 79,110
900 data 108,110,137,110
910 data 167,110,196,110
920 data 225,110,254,110
930 data   0, 27,191,209
940 data   0,255,200,  0
950 data  21,121,129,255
960 data 255,  0,  0,255
970 data   7,  7,241,242
980 data 243,  2,  7,  6
990 data   6,  6,  6,  6
1000 data   6,  6,  6,255
1010 data 252
1020 rem ---------------
1030 data  50,160, 79,160
1040 data 108,160,137,160
1050 data 167,160,196,160
1060 data 225,160,254,160
1070 data   0, 27,240,209
1080 data   0,255,200,  0
1090 data  21,121,129,255
1100 data 255,  0,  0,255
1110 data   4,  4,241,242
```

```
1120 data 243,  2,  7,  6
1130 data   6,  6,  6,  6
1140 data   6,  6,  6,255
1150 data 252
1160 rem ---------------
1170 data  50,210, 79,210
1180 data 108,210,137,210
1190 data 167,210,196,210
1200 data 225,210,254,210
1210 data   0, 27,240,209
1220 data   0,255,200,  0
1230 data  21,121,129,255
1240 data 255,  0,  0,255
1250 data   5,  5,241,242
1260 data 243,  2,  7,  6
1270 data   6,  6,  6,  6
1280 data   6,  6,  6,255
1290 data 252
1300 :
1310 rem *********************
1320 rem *   sprite-data's   *
1330 rem *********************
1340 :
1350 data   0,  0,  0
1360 data   0,  0,  0
1370 data   0,  0,  0
1380 data   0,  0,  0
1390 data   0,  0,  0
1400 data   0,  0,  0
1410 data   0,  0,  0
1420 data   0,  0,  0
1430 data   0,  0,  0
1440 data   0,  0,  0
1450 data   0,  0,  2
1460 data   0,  0, 10
1470 data   0,  0, 42
1480 data   0,192,170
1490 data   3, 50,170
1500 data  10,170,160
1510 data  42,170,165
```

```
1520 data 170,170,175
1530 data 171,255,229
1540 data  42,170,160
1550 data  10,170,170
```

2.5.2 Speed up mit dem VIC II

Doch, Sie haben richtig gelesen Speed up mit dem VIC. Mit dieser Überschrift ist gemeint, mittels des Videocontrollers BASIC- und Maschinenprogramme bis zu 25% schneller ablaufen zu lassen, ohne jedoch dabei auf den 40-Zeichenschirm verzichten zu müssen. Die Lösung dies Problems stellt natürlich wieder einmal der Rasterinterrupt dar, mit dem man fast alles, was mit Grafik zu tun hat, programmieren kann. Das Prinzip des Speed up Programms ist folgendes:

1. Initialisieren des Rasterinterrupts

2. Auslösen eines Interrupts am unteren Ende des Textfensters

3. Aktivieren des 2 MHZ-Modus durch Setzen von Bit 0 im Register 48

4. Initialisieren des Rasterinterrupts

5. Auslösen eines Interrupts am oberen Anfang des Textschirms

6. Zurücknehmen des Systemtaktes von 2 auf 1 MHZ

7. Wieder zu 1.

Wie Sie sehen ist das Prinzip relativ einfach. Doch werden Sie jetzt vielleciht fragen, wieso an dieser Stelle noch ein spezielles Speed up Programm steht, obwohl doch dieses Speed up genauso gut mit dem VIC-Simulationsprogramm zu erreichen gewesen wäre und auch zu erreichen ist. Nun, aufgrund der verlängerten IRQ-Routine beim VIC-Simulationsprogramm, dauert die

Abarbeitung der IRQ-Routine länger, d. h., das 2 MHZ-Flag wird später gesetzt. Dieses wird nun beim Speed up Programm umgangen, denn die einzigen VIC-Register die im Interrupt beeinflusst werden müssen, sind das Register 48, das Register 17 und das Register 18. Daher können wir mit dem Speed up Programm eine 25%ige Geschwindigkeitserhöhung und mit dem VIC-Simulationsprogramm nur eine 22%ige Geschwindigkeitserhöhung erreichen. Ein Tip: Vor der Ausführung jeglicher I/O Operationen sollte das Speed up Programm mit RUN-STOP/RESTORE ausgeschaltet werden, um Timingprobleme zu umgehen. Flimmert der Bildschirm nach dieser Tastenkombination merkwürdig, so liegt das daran, daß das 2 MHZ Flag noch gesetzt ist. Geben Sie dann bitte folgendes ein:

POKE 53248+48,0

Hierdurch wird der Systemtakt auf 1 MHZ heruntergenommen, und der Videocontroller ist wieder in der Lage zu arbeiten. Hier nun als Erstes das Assemblerlisting und dann der Basiclader. Das Programm muß im 64er-Modus eingegeben und mit 'SYS 49152' aktiviert werden.

Assemblerlisting:

```
profi-ass 64 v2.0

110:   c000                  .opt p1,oo
120:              ;
130:              ;***********************
140:              ;*  speed up program *
150:              ;***********************
160:              ;
170:   c000                  *=   $c000
180:              ;
190:   0032       zolow   =   50      ; low-byte oben
200:   0000       zohigh  =   0       ; high-byte oben
210:   00fa       zulow   =   250     ;  low-byte unten
220:   0000       zuhigh  =   0       ; high-byte unten
230:              ;
```

```
240:   d000          v       =     53248   ; basisadresse vic
250:                 ;
260:                 ;*********************************
270:                 ;* initialisieren des raster-irq *
280:                 ;*********************************
290:                 ;
300:   c000 78       init    sei             ; interrupt verhindern
310:   c001 a9 2b            lda   #<neuirq  ; low-byte neuirq
320:   c003 a0 c0            ldy   #>neuirq  ; high byte neuirq
330:   c005 8d 14 03         sta   $0314     ; in irq-vektor low
340:   c008 8c 15 03         sty   $0315     ; in irq-vektor high
350:   c00b a9 32            lda   #zolow    ; obere raster-zeile
360:   c00d 8d 12 d0         sta   v+18      ; low-byte
370:   c010 ad 11 d0         lda   v+17      ; reg.17 lesen
380:   c013 29 7f            and   #%01111111  ; bit 7 loeschen
390:   c015 09 00            ora   #zohigh   ; high-byte
400:   c017 8d 11 d0         sta   v+17      ; reg.17 wieder schreiben
410:   c01a a9 01            lda   #1        ; flag naechster irq unten
420:   c01c 85 02            sta   2         ; in 2
430:   c01e a9 81            lda   #%10000001  ; maske raster-irq
440:   c020 8d 1a d0         sta   v+26      ; ins imr schreiben
450:   c023 ad 19 d0         lda   v+25      ; irr auslesen
460:   c026 8d 19 d0         sta   v+25      ; und loeschen
470:   c029 58               cli             ; irq-freigeben
480:   c02a 60               rts             ; zum basic
490:                 ;
500:                 ;************************
510:                 ;*   neue irq-routine   *
520:                 ;************************
530:                 ;
540:   c02b ad 19 d0 neuirq  lda   v+25      ; irr auslesen
550:   c02e 8d 19 d0         sta   v+25      ; und loeschen
560:   c031 30 07            bmi   raster    ; bei raster-irq sprung
570:   c033 ad 0d dc         lda   $dc0d     ; irr cia1 loeschen
580:   c036 58               cli             ; interrupt freigeben
590:   c037 4c 31 ea         jmp   $ea31     ; und zum alten irq
600:   c03a a5 02    raster  lda   2         ; flag in akku
610:   c03c 49 01            eor   #1        ; umdrehen
620:   c03e 85 02            sta   2         ; wieder als falg
630:   c040 8d 30 d0         sta   v+48      ; 1/2 mhz-flag
```

```
640:    c043 f0 12              beq  label1  ; bei 1 mhz sprung
650:    c045 a9 32              lda  #zolow  ; zeile low
660:    c047 8d 12 d0           sta  v+18    ; als rasterzeile low
670:    c04a ad 11 d0           lda  v+17    ; reg.17 auslesen
680:    c04d 29 7f              and  #%01111111  ; bit 7 loeschen
690:    c04f 09 00              ora  #zohigh  ; high-byte setzen
700:    c051 8d 11 d0           sta  v+17    ; reg.17 schreiben
710:    c054 4c 66 c0           jmp  label2  ; ok zum ende
720:    c057 a9 fa    label1    lda  #zulow  ; zeile low
730:    c059 8d 12 d0           sta  v+18    ; low-byte rasterzeile
740:    c05c ad 11 d0           lda  v+17    ; reg.17 auslesen
750:    c05f 29 7f              and  #%01111111  ; bit 7 loeschen
760:    c061 09 00              ora  #zuhigh  ; high-byte oren
770:    c063 8d 11 d0           sta  v+17    ; reg.17 setzen
780:    c066 4c bc fe label2    jmp  $febc   ; irq-ende
```

Basiclader:

```
100 rem *****************
110 rem *   speed up    *
120 rem *  basiclader   *
130 rem *****************
140 :
150 for i=49152 to 49256
160 :   read a
170 :   s=s+a
180 :   poke i,a
190 next i
200 :
210 if s<>11342 then print "fehler in data's!!!" : end
220 :
230 print "programm ok!"
240 :
250 end
260 :
270 data120,169,043,160,192,141,020
280 data003,140,021,003,169,050,141
290 data018,208,173,017,208,041,127
300 data009,000,141,017,208,169,001
```

```
310 data133,002,169,129,141,026,208
320 data173,025,208,141,025,208,088
330 data096,173,025,208,141,025,208
340 data048,007,173,013,220,088,076
350 data049,234,165,002,073,001,133
360 data002,141,048,208,240,018,169
370 data050,141,018,208,173,017,208
380 data041,127,009,000,141,017,208
390 data076,102,192,169,250,141,018
400 data208,173,017,208,041,127,009
410 data000,141,017,208,076,188,254
```

2.6 Scrolling

Scrolling. Durch dieses einfache Wort können Sie den Bildschirm Ihres C128 beliebig erweitern. Scrolling bedeutet daß beliebige Teile des Bildschirm weggerollt oder wieder hineingerollt werden. Auch ein Scrolling mit Umlauf ist möglich. In der Computergrafik unterscheidet man zwischen zwei verschiedenen Arten des Scrollings:

1. Smooth Scrolling ('Weich' Scrolling)
2. Normal Scrolling (Zeichenweise)

Hierbei stellt uns die erste Art des Scrollings vor ernste Probleme, da es zwar theoretisch relativ einfach zu verwirklichen ist, aber rein zeitmäßig weit hinter unseren Anspruchen hinterherhinkt. Aus diesem Grund wird an dieser Stelle nur eines der Prinzipien erklärt.
Als Grundvoraussetzung sollten Sie über Rasterzeilen, die IRQ-Möglichkeiten des VIC's und die VIC-Register 17 und 22 Bescheid wissen. Wie Sie nun also wissen sollten, ist es mittels der VIC-Register 17 und 22 möglich, den Bildschirm rasterpunkt- bzw. rasterzeilenweise zu verschieben. Geschieht dies fließend hintereinander, so erhalten wir ein weiches Scrolling, das sogenannte Smooth Scrolling. Problematisch wird es allerdings erst, wenn wir alle acht Rasterpunkte bzw. Rasterzeilen

ausgenutzt haben, es also mittels der Register 17 oder 22 nicht mehr 'weiter' geht. Es bleibt uns dann nichts weiter übrig, als das entsprechende Register wieder zurückzusetzen und den ganzen Bildschirm um eine Spalte oder Zeile nach oben, unten, links oder rechts zu scrollen. Und eben hier liegt das Problem: Ein wirklich gleichmäßiges Scrolling erreichen wir nur, wenn wir mit dem Verschieben beginnen, wenn der VIC gerade die unterste Rasterzeile (251) des Textfensters aufbaut und wieder fertig sind, wenn der Elektrodenstrahl wieder an der oberen Bildschirmfensterkante ist. Dies haben wir trotz eines tiefen Griffs in die Trickkiste (FAST-Modus, illegale Opcodes etc.) nur unvollständig erreichen können. Aus diesem Grunde hier ein kleines Basicdemo, bei dem ein Flimmern natürlich nicht absolut auszuschließen ist:

Basicdemo:

```
100 rem ****************************
110 rem *   smooth-scroll-demo    *
120 rem ****************************
130 :
140 scnclr
150 :
160 list
170 :
180 v = 53248
190 :
200 poke v+17,peek(v+17) and not 8
210 :
220 for i=7 to 0 step -1
230 :   poke v+17,(peek (v+17) and 248) or i
231 sleep .5
240 next i
250 :
260 print
270 :
280 goto 220
```

Die zweite Form des Scrollings ist das zeichenweise Scrolling. Dies wird sogar vom Commodore Betriebssystem mittels der Windowtechnik (teilweise) unterstützt. Teilweise deshalb, weil es nur das Auf- und Abwärtsscrolling unterstützt und dies nicht einmal vollständig. So ist es beispielsweise nur mit einigen Rumkopiererreihen möglich, das Scrolling mit Umlauf zu ermöglichen.

2.7 Der Lightpen

Am Anfang der Technisierung bestand ein vollständiger Computerarbeitplatz aus einer Tastatur, einem Bildschirm, einem Speichermedium und eventuel auch einem Drucker. Die Tastatur war also das einzigste Eingabemedium. Doch im Laufe der Zeit kamen immer neue Eingabemedien hinzu. Diese neuen Eingabenmedien reichten von der Menuesteuerung mittels Joystick oder ählichem über manuelle Eingabeelemente wie z.B. Zeichenplatten bis zu Bildschirmen auf denen mittels eines Tastendrucks Befehle ausgelöst werden konnten.

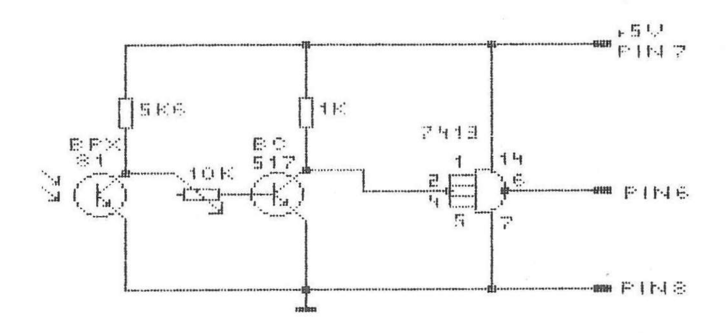
Im Laufe dieser Entwicklung stand auch der Lightpen, der als komfortabeles wie auch preiswertes Eingabemedium neben der Maus immer mehr an Bedeutung gewinnt. Aus diesem Grund wollen wir Ihnen in diesem Buch eine komplette Bauanleitung für einen Lightpen vorstellen. Die Idee und der Schaltplan für den Lightpen wurde mit freundlicher Genehmigung von Jürgen Steigers aus seinem Beitrag in der *DATA WELT* 3/85 entnommen. Dieser Lightpen ist zwar nicht unbedingt so qualitativ hochwertig wie ein käuflicher, erfüllt aber für unsere Anwendungen seinen Zweck. Auf die Hardwarebeschreibung wollen wir nicht weiter eingehen, da sie unserer Meinung nach an dieser Stelle nichts zu suchen hat. Wir geben also nur den Schaltplan, die Stückliste und das Layout an. Die Schaltung ist so simpel, daß sie auch für den absoluten Anfänger kein Problem darstellen sollte. Allerdings sollte noch das Prinzip eines Lightpens erklärt werden. Das Prinzip eines Lightpens hängt direkt mit dem Aufbau des Bildes zusammen. Wie wir wissen,

wird das Bild Punkt für Punkt und Zeile für Zeile durch einen Elektrodenstrahl aufgebaut. Durchläuft der Strahl nun die Koordinaten, auf die der Lightpen zeigt, so erkennt dieser dies an einer erhöhten Leuchtintensität und gibt das Signal an den Videocontroller weiter. Da der VIC nun weiß, welcher Punkt gerade aufgebaut wird, ist er in der Lage, die Koordinaten zu berechnen, auf die der Lichtgriffel zeigt. Nachdem wir nun wissen, wie ein Lightpen theoretisch funktioniert, hier die Bauanleitung:

Sie benötigen:

a) Fototransistor BPX 81
b) Transistor BC 517
c) Schmittrigger SN7413
d) Widerstand 5K6
e) Widerstand 1K
f) Potentiometer 10K
g) Controlportstecker
h) Platine
i) Ausgedienten Kugelschreiber
j) ca. 1m Kabel 2adrig

Schaltplan:



Haben Sie nun den Lightpen anhand des oben stehenden Schaltplans aufgebaut, so können Sie ihn mit folgendem Programm testen, das ein Beispiel für die Menuesteuerung mittels des Lightpens darstellt.

Demoprogramm:

```
100 rem ********************************
110 rem *   light-pen menueauswahl    *
120 rem ********************************
130 :
140 color 0,1 : color 4,1 : color 5,2
150 :
160 scnclr : dim a$(20)
170 :
180 for i=1 to 6
190 :   read a$(i)
200 :   x(i)=20-len(a$(i))/2
210 :   y(i)=i*2
220 :   char 1,x(i),y(i),a$(i)
230 next i
240 :
250 print chr$(19);chr$(19);
260 x=(pen(0)/8)-12
270 y=(pen(1)/8)- 6
280 :
290 if x<0 or y<0 or x>39 or y>24 then 370
300 i=y/2 : if i=li then 250
310 :
320 char 1,x(li),y(li),"Æ"+a$(li)
330 char 1,x(i),y(i),""+a$(i)
340 :
350 li=i
360 :
370 get a$ : if a$<>chr$(13) then 260
380 :
390 window 0,13,39,24,1
400 :
410 print
```

```
420 :
430 on i goto 470,530,590,670,770,830
440 :
450 goto 250
460 :
470 rem *** catalog ***
480 :
490 catalog u8
500 :
510 goto 250
520 :
530 rem *** disk-status ***
540 :
550 print ds$
560 :
570 goto 250
580 :
590 rem *** scratch ***
600 :
610 input "name";na$
620 :
630 scratch (na$),u8
640 :
650 goto 250
660 :
670 rem *** rename ***
680 :
690 input "alter name";an$
700 :
710 input "neue  name";nn$
720 :
730 rename (an$) to (nn$),u8
740 :
750 goto 250
760 :
770 rem *** collect ***
780 :
790 collect
800 :
810 goto 250
```

```
820 :
830 rem *** concat ***
840 :
850 input "quellfile";qn$
860 :
870 input "zielfile ";zn$
880 :
890 concat (qn$) to (zn$)
900 :
910 goto 250
920 :
930 rem *******************
940 rem *   menuepunkte    *
950 rem *******************
960 :
970 data catalog
980 data disk-status
990 data scratch
1000 data rename
1010 data collect
1020 data concat
```

Wie Sie den Basicbefehl PEN anwenden können, erfahren Sie in den Kapiteln 1.3ff.

2.8 Das Farb-RAM

Eigentlich ist es sinnlos, etwas über das sogenannte Farb-RAM zu schreiben, denn man könnte alle wesentlichen Fakten in einem Satz zusammenfassen. Dies wollen wir jedoch nicht tun, da wir nicht davon ausgehen können, daß auch Computeranfänger dieses Fachchinesisch verstehen würden. Hier also ausführlich das, was es über das sogenannte Farb-RAM zu sagen gibt:

Das sogenannte Farb-RAM ist kein RAM-Baustein, wie er im Prozessorspeicher benutzt wird, sondern er wird direkt vom VIC angesprochen. Nun haben Sie aber vielleicht schon festgestellt, daß beim Auslesen einer Farb-RAM-Adresse sich immer Werte über 128 ergeben. Dies liegt daran, daß das Farb-RAM nicht 8192 Speicherplätze, sondern nur 4096 Speicherplätze belegt. Diese 4096 Speicherplätze wiederum sind in 1024 Teilstücke zu je vier Bits unterteil. Die oberen acht Bits gelten bei einem Lesezugriff als gesetzt. Diese speicherplatzsparende Kodierung der Farbe mit nur vier Bits liegt darin begründet, daß zur Darstellung aller 16 Farben nur maximal vier Bits benötigt werden. Und zu guter Letzt: Der Farb-RAM ist nicht verschiebbar und liegt immer bei 55296 ($D800) - dies ist, wie oben schon gesagt, darin begründet, daß das Farb-RAM direkt an den VIC angeschlossen ist und daher nicht durch den Adreßmanager oder die MMU verschoben werden kann.

III. Der VDC (8563)

3.1 Der VDC

Dieses große Kapitel war ursprünglich als kleiner Unterabschnitt in unserem Hardwarekapitel geplant. Als wir jedoch die Registerbeschreibung des 8566 zum ersten Mal in die Hände bekamen, da wußten wir, daß wir es bei diesem kleinen Abschnitt nicht belassen konnten.
Der Video-Device-Controller ist bestimmt eines der Bauteile, wenn nicht sogar das Bauteil, das den Commodore 128 ein ganzes Stück weiter zu der Klasse der Personal-Computer bringt. Der Hauptunterschied, den der VDC gegenüber dem VIC II aufweist, ist natürlich die 80-Zeichendarstellung. Aber es gibt weitere Unterschiede, die meistens, jedoch nicht immer, den VDC als den besseren Videocontroller hinstellen. Fangen wir zunächst, mit den scheinbaren Schwachpunkten an:

Zunächst, man konnte es schon oft in vielen Zeitschriften lesen, wird dem VDC das Fehlen von Sprites angelastet. Es stellt sich nun die Frage, in wieweit in einer Textverarbeitung, bei Kalkulations- oder Datenverwaltungsprogrammen Sprites notwendig sind.

Der zweite Grund ist die Tatsache, daß immer noch Gerüchte herumirren, der VDC wäre nicht voll grafikfähig. Um diese, bestimmt interessante Tatsache vorwegzunehmen: Der VDC ist grafikfähig - und das sogar sehr gut, mit seiner Auflösung von 640*200 Punkten.

Der dritte und letzte mir präsente Grund ist die bei Kritikern verrufene schlechte Speicherverwaltung des eigenen Speichers. Zugegeben, die schnellste Methode der Speicherplatzverwaltung kennt der VDC nicht gerade, jedoch eine der gängigsten, mit denen in unserem Fall 16384 Bytes von Hauptspeicher unabhängig (!) verwaltet werden können.

Doch nun genug der negativen Kritik, wenden wir uns jetzt den positiven Dingen des Video-Device-Controlers zu.

Primärer Grund bleibt, war und wird die Möglichkeit der 80-ZeichenDarstellung sein, die eine vernünftige Textverarbeitung z. B. erst möglich macht.

Ein weiteres Feature des VDC ist seine Möglichkeit, 4 (in Worten: -vier-) verschiedene Zeichensätze zu je 128 Zeichen exclusive (!) reverse (die kann er alleine darstellen) auf einmal darzustellen. Weiterhin, ich möchte das nicht als Einzelpunkt aufzählen, obwohl es mit Sicherheit gerechtfertigt wäre, kann er ohne unser Zutun alle Zeichen zusätzlich noch unterstreichen oder blinkend darstellen - oder auch beides gleichzeitig machen (man denke nun wieder an eine Textverarbeitung...).

Sämtliche Zeichen stehen im Ram des VDC, so daß eine Änderung überhaupt kein Problem mehr darstellt. Sie können es erkennen, wenn Sie die Zeichensätze durch die ASCII/DIN-Taste vertauschen.

Weiterhin verfügt der VDC über soviel eigenes RAM, daß er in der Lage ist, zwei verschiedene, farbige Bildschirmseiten zu verwalten.

Der VDC verfügt über sehr gute Speicherverwaltungshilfen, mit denen trotz der relativ komplexen Adressierung gegenüber dem VIC II, eine schnelle Verarbeitungsgeschwindigkeit möglich wird.

Zu guter letzt ist, wie schon gesagt, der VDC voll grafikfähig.

Nun aber zum praktischen Teil. Zunächst wollen wir Ihnen die Registerbeschreibung des 8566 im folgenden beschreiben. Diese soll Ihnen später als praktische Nachschlagetabelle dienen, doch zuerst einmal einen weiteren Eindruck vermitteln, welche Möglichkeiten sich durch diesen neuen Video-Controller auftun.

3.2 Registerbeschreibung des 8546

Registernummer Beschreibung

Reg. 00................ horizontal total; in diesem Register wird die Anzahl der Zeichen pro Zeile einschließlich dem Strahlrücklauf angegeben. (126)

Reg. 01................ horizontal displayed; dieses Register beinhaltet den Wert der tatsächlichen Anzahl der Buchstaben pro Zeile. Der angegebene Wert muß kleiner sein, als der in Register 0.

Reg. 02................ horizontal sync position; dieses Register ist für die Synchronisation des linken Randes zuständig. Alle Werte kleiner als die im Register 0 sind erlaubt. Wird der Wert im Register vergrößert, so bewegt sich der linke Rand nach links, wird er verkleinert, entsprechend nach rechts. (102)

Reg. 03................ sync width;
.... Bits 0-3 bestimmen die horizontale Sync-Puls-Breite in Zeilen. Ein Null-Wert kann nicht programmiert werden.
.... Bits 4-7 bestimmen die vertikale Sync-Puls-Breite im Vielfachen eines Rasterdurchlaufes. Der Wert 0 entspricht dem Wert 16. (73)

Reg. 04................ vertikal total; dieses Register beinhaltet die Anzahl der totalen Zeilen einschließlich des vertikalen Strahlenrücklaufs. (39)

Reg. 05................ vertikal total adjust;
.... Bits 0-4 dienen der Feineinstellung des Registers 4
.... Bits 5-6 sind immer gesetzt. (224)

Reg. 06............... vertikal displayed; in diesem Register wird die Anzahl der darzustellenden Zeilen festgelegt (ähnlich Register 1). Jeder Wert kleiner als der im Register 4 ist möglich. (25)

Reg. 07............... vertikal sync position; dieses Register (ähnlich Register 2) setzt den oberen Rand des Bildschirms. Bei Vergrößerung des Wertes verschiebt sich der Bildschirm nach oben, wird er verkleinert, verschiebt er sich entsprechend nach unten. (32)

Reg. 08............... interlace mode;
.... Bits 0 und 1 bestimmen den Abtastmodus.
.... 00 und 01= Non-Interlace-Mode
.... (Bild scheint zu flackern)
.... 11=Interlace-Sync & Video-Mode (252)

Reg. 09............... character total vertikal;
.... Bits 0-4 bestimmen die vertikale Rasterzeilenanzahl minus 1;
.... Bits 5-7 sind immer gesetzt. (231)

Reg. 10............... cursor mode/start raster;
.... Bits 0-4 bestimmen die Startrasterzeile des Cursors
.... Bits 5-6 sind für das Cursorblinken zuständig:
.... 00 = Cursor blinkt nicht 01 = Cursor wird
.... gar nicht angezeigt
.... 10 = Cursor blinkt schnell
.... 11 = Cursor blinkt normal (160)

Reg. 11............... cursor end scan line;
.... Bits 0-4 bestimmen die Rasterzeile, in der der Cursor endet. Nur diese 4 Bits sind relevant. (213)

Reg. 12............... displayed start adress (high); in diesem Register wird das High-Byte des Video-Ram-Anfangs gespeichert (standardmäßig bei $0000)

Reg. 13 .... displayed start adress (low); entsprechend Register 12 wird hier das Low-Byte gespeichert.

Reg. 14/15 .... cursor position (high/low); da der VDC den Cursor hardwaremäßig blinken läßt, wird in diesem Register die momentane Adresse des Cursors übergeben. Diese Adresse läßt sich mit der Formel "ad=x+80*y" errechnen.

Reg. 16 .... lightpen vertikal; dieses Register gibt die vertikale Adresse des Lightpens wieder. Es kann nur gelesen werden. Nur die unteren Bits sind relevant. Bits 6 und 7 sind immer null.

Reg. 17 .... lightpen horizontal; entsprechend Register 16 kann aus diesem Register die horizontale Position des Lightpens ermittelt werden. Die Einheit ist jeweils ein Zeichen groß.

Reg. 18/19 .... updated adress; wenn Daten in den VDC, der ja bekannlich über einen eigenen 16K großen Speicher verfügt, geschrieben oder aus diesem gelesen werden, so muß in diesem Register die Adresse (0-16383; $0000-$3FFF) der entsprechenden Speicherstelle übergeben werden.

Reg. 20/21 .... attribute adress (high/low); diese beiden Register definieren die Startadresse des Attribut-Rams. (Standard: 2048; $0800)

Reg. 22 .... character total/character displayed;
.... Bits 0-3 bestimmen die Anzahl der dargestellten Horizontalzeilen.
.... Bits 4-7 bestimmen die Totalanzahl der dargestellten Horizontalzeilen. (120)

Reg. 23 .... lines displayed vertikal; dieses Register bestimmt die Anzahl der dargestellten Vertikalzeilen.

Reg. 24............... control register 1;

.... Bits 0-4: Mit diesen Bits kann der vertikale Rand in Rasterzeilen verschoben werden.
.... Bit 5: Bedeutung leider noch unbekannt.
.... Bit 6: Wird dieses Bit gesetzt, so wird der gesamt Bildschirm revers dargestellt.
.... Bit 7: Wird dieses Bit gesetzt, so wird bei Beschreiben des Registers 30 der Bereich von Block-Startadresse bis zur Up-Date Adresse kopiert. Bei gelöschtem Bit wird die Update-Adresse mit dem Data-Register gefüllt.

Reg. 25............... control register 2;
....
.... Bits 0-3: Mit diesen Bits kann der horizontale Rand in Rasterzeilen verschoben werden.
.... Smoothscrolling)
.... Bit 4: Bei gesetztem 4. Bit erscheinen die Zeichen in doppelter Breite.
.... Bit 5: Bei gesetztem 5. Bit wird der eventuell bestehende Zwischenraum zweier Zeichen in der Farbe des Zeichens, das zuletzt dargestellt wurde, aufgefüllt.
.... Bit 6: Dieses Bit gibt an, ob die Farbe der einzelnen Zeichen aus dem Attribute-Ram geholt werden soll oder ob alle Zeichen monochrom, mit der bestimmten Farbe in Register 26 (s.u.) erscheinen sollen (Bit=0)
Bit 7: Ist dieses Bit gesetzt, so wird von der Textdarstellung auf die hochaufgelöste Darstellung umgeschgaltet. (71)

Reg. 26............... foreground/background;
.... Bits 0-3 bestimmen die Hintergrundfarbe
.... Bits 4-7 bestimmen die Zeichenfarbe

Reg. 27................ adress increment row; in diesem Register wird die Anzahl der Bytes festgelegt, die zum Video-Ram in jeder Spalte zu addieren sind

Reg. 28................ character basis adress;
.... Bit 4 betimmt den Ram-Typ.
.... Bits 5-7 bestimmen die Basis des Zeichengenerators. Sie entsprechen den Adressbits 13-15. Hierdurch kann der Zeichengenerator in 8K-Schritten verschoben werden. In diesem Fall ist nur Bit 5 relevant, da der VDC nur über 16K-Ram verfügt.

Reg. 29................ underline scan line;
.... Bits 0-4 geben die Rasterzeile an, in denen die Zeichen unter- bzw. durchgestrichen gezeichnet werden sollen.

Reg. 30................ word count; in dieses Register schreibt man die Anzahl der Zeichen, die an die Up-Date-Adresse geschrieben werden sollen.
.... Bei gesetztem Copy-Bit (siehe Reg. 24) wird hier die Anzahl der zu kopierenden Zeichen angegeben.

Reg. 31................ data; in dieses Register schreibt man den Wert, den man an die momentane Up-Date Adresse des VDC-Rams schreiben will.
.... Wird dieses Register gelesen, so entspricht der gelesene Wert dem im VDC-Ram, auf den die Register 18 & 19 den Zeiger bilden.

Reg. 32/33.......... block-Start-Adress (high/low); in dieses Register wird die Startadresse eines zu kopierenden Blocks geschrieben.

Reg. 34............... display enable begin; in dieses Register wird die Anzahl der Zeichen von Beginn der dargestellten Zeichen bis zur positiven Flanke des Display-eneable-Pins geschrieben.(125)

Reg. 35............... display enable end; wie Register 34, jedoch bis zur negativen Flanke.

Reg. 36............... d-ram refresh rate
.... Bits 0-4 geben die Rate an, mit der der Speicher des VDC aufgefrischt werden muß

Der nun folgende Abschnitt behandelt einige Grundsätze über den VD-Controller, die wirklich **SEHR WICHTIG** sind, da erst sie das Arbeiten mit dem VDC ermöglichen.

3.3 Grundsätzliches zum VDC

Zunächst wollen wir uns einer Besonderheit des VDCs widmen, die unsere Arbeit in Zukunpft bestimmen wird. Wir können die Register nicht durch POKE, wie beim VIC, adressieren. Um einen Wert in ein bestimmtes Register zu schreiben oder aus einem Register zu lesen, ist ein spezielles Verfahren notwendig. Hierzu dienen die Register 0 und 1. Nur diese beiden Register können direkt angesprochen werden. Im einzelnen sind jeweils folgende Arbeitsschritte notwendig:

Schreiben eines Bytes in ein bestimmtes Register

1) Die Registernummer wird in das Register 0 ($D600) des VDC geschrieben.
2) Nun muß solage gewartet werden, bis sich der VDC durch Setzen des 7. Bits dazu bereit erklärt, weitere Arbeiten vorzunehmen.

3) Als nächstes wird der Wert, der in das Register geschrieben werden soll, im Register 1 des VDC abgelegt.
4) Theoretisch müßte sich unser Wert nun im vorgesehenen Register befinden, und Sie müßten das anhand einer Bildschirmveränderung erkennen.

Lesen eines Bytes aus einem bestimmten Register

1) Die Registernummer wird, genauso wie beim Schreiben eines VDC-Registers, in das Register 0 des VDC geschrieben.

2) Wiederum muß auf das Setzen des 7. Bits gewartet werden.

3) Nun kann Register 1 gelesen werden, das den Wert des entsprechenden Registers enthält.

Das zwischen Theorie und Praxis jedoch ein weiter Abgrund klafft, zeigt das folgedne Problem, das sich nun ergibt:
Es ist von BASIC aus nicht immer möglich, diese POKErei so schnell vorzunehmen, das sich der VDC damit zufrieden gibt. Er reagiert, wenn wir ihm zu langsam sind, indem er unsere Daten verschluckt, zumindest aber verfälscht. Um dieses zu vermeiden, müssen wir, ob wir wollen oder nicht, die Maschinensprache hinzuziehen.

Im anschließenden Kapitel stellen wir Ihnen deswegen ein Programm vor, das die folgenden zwei Hauptaufgaben bewältigen kann.

1) Lesen und Beschreiben eines Registers
2) Lesen und Beschreiben des VDC eigenen Rams.

3.4 VDC Utilities

Hier ist nun zunächst das angekündigte Programm in Form eines Assemblerlistings. Wie Sie hier selber sehen können, wurde das Programm mit dem Assembler PROFI-MAT erstellt. Sollten Sie den Assembler/Monitor PROFI-MAT besitzen, so sollten Sie folgendes beachten: Erstellen und assemblieren Sie Ihr Programm grundsätzlich im 64er Modus, da das 128er Betriebssystem nicht mit dem 64er Betriebssystem kompatibel ist.

Sie selber können natürlich dieses Programm auch mit dem eingebauten Monitor des Commodore 128 eingeben. Weiter unten ist jedoch ein sehr komfortabler BASIC-Lader abgedruckt, um auch Programmierern, die der Assemblersprache nicht mächtig sind, das Arbeiten mit diesen Utilities möglich zu machen.

3.4.1 Das Assembler Listing

```
110:  0c00                      .opt p
120:  0c00                      .sym 2
130:  0c00                      *=   $0c00
140:                     ;
150:                     ;*********************************
160:                     ;* speichern eines wertes       *
170:                     ;* in ein bestimmtes register *
180:                     ;* registernr. im akku          *
190:                     ;* wert im x-register           *
200:                     ;*********************************
210:                     ;
220:  0c00 8d 00 d6 strreg  sta  $d600    ;reg.nr. im akku
230:  0c03 2c 00 d6 loop1   bit  $d600    ;warten bis
240:  0c06 10 fb            bpl  loop1    ;reg.nr. akzeptiert,
250:  0c08 8e 01 d6         stx  $d601    ;dann x-reg. abspeichern
260:  0c0b 60               rts
270:                     ;
```

```
280:                    ;*******************************************
290:                    ;* lesen eines wertes aus                  *
300:                    ;* einem bestimmten register               *
310:                    ;* registernr. im akku                     *
320:                    ;* wert wird im akku uebergeben, und       *
330:                    ;* kann dann mit dem "rreg" basic-befehl   *
340:                    ;* gelesen werden                          *
350:                    ;*******************************************
360:                    ;
370:    0c0c 8d 00 d6 getreg   sta  $d600   ;reg.nr. im akku
380:    0c0f 2c 00 d6 loop2    bit  $d600   ;waten bis
390:    0c12 10 fb             bpl  loop2   ;reg.nr. akzeptiert
400:    0c14 ad 01 d6          lda  $d601   ;dann akku mit entsprechendem
410:    0c17 a2 00             ldx  #$00    ;wert laden, und den rest
420:    0c19 a0 00             ldy  #$00    ;loeschen
430:    0c1b 60                rts
440:                    ;
450:                    ;*** setzen der up-date address ***
460:                    ;
470:                    ;*************************************
480:                    ;* low-byte im akku, high-byte in x *
490:                    ;*************************************
500:                    ;
510:    0c1c 48       setuda   pha          ;akku retten
520:    0c1d a9 12              lda  #18     ;reg.nr. fuer high-byte set-
zen
530:    0c1f 20 00 0c          jsr  strreg
540:    0c22 68                pla          ;low-byte ins x-register
550:    0c23 aa                tax          ;bringen, und in reg.19
560:    0c24 a9 13             lda  #19     ;reg.nr. fuer low-byte setzen
570:    0c26 4c 00 0c          jmp  strreg  ;speichern
580:                    ;
590:                    ;*** schreiben in das vdc-ram ***
600:                    ;
610:                    ;**************************************
620:                    ;low-byte der addresse im x-register *
630:                    ;high-byte der addresse im akku      *
640:                    ;wert im y-register                  *
650:                    ;**************************************
660:                    ;
```

```
670:   0c29 20 1c 0c pokeram  jsr  setuda  ;up-date adress setzen
680:   0c2c 98                tya          ;y-register ins
690:   0c2d aa                tax          ;x-register bringen
700:   0c2e a9 1f             lda  #31     ;und im data-register
710:   0c30 4c 00 0c          jmp  strreg  ;abspeichern
720:                 ;
730:                 ;*** lesen aus dem vdc-ram ***
740:                 ;
750:                 ;*****************************************
760:                 ;* addresse muss im low/high format     *
770:                 ;* in x und a uebergeben werden          *
780:                 ;* der gelesene wert befindet sich       *
790:                 ;* hinterher im akku  und kann mit dem *
800:                 ;* "rreg" basic-befehl ermittelt werden *
810:                 ;*****************************************
820:                 ;
830:   0c33 20 1c 0c peekram  jsr  setuda  ;up-date address setzen
840:   0c36 a9 1f             lda  #31     ;data register
850:   0c38 4c 0c 0c          jmp  getreg  ;auslesen
```

```
Symboltabelle:
peekram   0c33   pokeram   0c29
setuda    0c1c   loop2     0c0f
getreg    0c0c   loop1     0c03
strreg    0c00
7 Symbole definiert
```

3.4.2 Der BASIC-Lader

```
10 rem ************************************
20 rem **                                **
30 rem **  vdc-utilities (basic-lader)  **
40 rem **  ---------------------------   **
50 rem ************************************
60 :
```

```
70 do while x<59
80 :read a$
90 :a=dec(a$)
100 :poke 3072+x,a
110 :ps=ps+a
120 :x=x+1
130 loop
140 :
150 if ps<>4926 then begin
160 :print chr$(7)
170 :print chr$(147)+chr$(2)+chr$(15);
180 :print "     fehler in datas      "
190 :end
200 bend
210 else:print chr$(147)+"datas ok"
220 :
230 print "programm abspeichern [j/n]"
240 getkey a$
250 if a$="n" then end
260 scnclr
270 input "programm-name";n$
280 print:print:print
290 print "disk-laufwerk bereitmachen und return druecken"
300 getkey a$:if a$<> chr$(13) then 300
310 bsave (n$),b0,p3072 to p3132
320 end
330 :
340 rem assembler-datas
350 :
360 data 8d,00,d6,2c,00,d6,10,fb,8e,01,d6,60,8d,00,d6,2c
370 data 00,d6,10,fb,ad,01,d6,a2,00,a0,00,60,48,a9,12,20
380 data 00,0c,68,aa,a9,13,4c,00,0c,20,1c,0c,98,aa,a9,1f
390 data 4c,00,0c,20,1c,0c,a9,1f,4c,0c,0c
```

3.4.3 Utilities - Erklärung

Hier sei nun die Funktionsweise unserer neuen Assembler-Routinen beschrieben:

1) Register beschreiben

Aufruf	SYS 3072 , registernummer , wert
Parameter	registernummer: Die Nummer des VDC-Registers, in das ein bestimmter Wert geschrieben werden soll. wert: Der Wert, der in dieses Register geschrieben werden soll
Beispiel	sys 3072 , 5 , 25

2) Register auslesen

Aufruf	sys 3084 , registernummer : rreg wert
Parameter	registernummer: Nummer des Registers, aus dem der Wert gelesen werden soll wert: Eine beliebige Variable, in der der gelesene Wert übergeben wird. Durch den RREG-Befehl werden dem übergeordneten BASIC-Programm diverse Übergabeparameter einer Assemblerroutine übergeben. Diese Parameter sind der Akkumulator, das X, das Y-Register und das Statusregister.

Beispiel sys 3084 , 25: rreg a:print a

3) Speicherstelle im VDC-Ram beschreiben

Aufruf SYS 3113 , low-Add , high-add , wert

Parameter low-add: Low-Byte der 16-Bit-Adresse, aus der der Wert gelesen werden soll. Das Low-Byte einer Adresse errechnet sich am besten mit folgender Formel:

lb = adresse - INT(adresse/256)*256

high-add: High-Byte einer Adresse. Zur Berechnung des High-Bytes eines 16-Bit-Wertes, eignet sich folgende Formel :

hb = INT(adresse/256)

wert: Der Wert, der an die bestimmte Adresse geschrieben werden soll.

Beispiel sys 3113,0,32,255

4) Speicherstelle aus dem VDC-Ram lesen

Aufruf SYS 3123,low-add,high-add:rreg wert

Parameter

low-add: Low-Byte der Adresse, aus der der Wert gelesen werden soll

high-add: High-Byte der Adresse, aus der der Wert gelesen werden soll

wert: beliebige Variable, in der der entsprechende Wert übergeben wird. Für RREG gilt das Obengesagte.

An einem einfachen Beispiel möchten wir Ihnen zu guter letzt den Umgang mit diesen neuen Routinen demonstrieren:

```
10 rem ***************************
20 rem ** bildschirm loeschen **
30 rem ***************************
40 :
50 for x=0 to 1999
60 :lb=x-int(x/256)*256
70 :hb=int(x/256)
80 :sys 3113,lb,hb,32
90 next x
```

Das ist natürlich nur ein ziemlich einfaches Beispiel, jedoch läßt sich gerade an solchen Beispielen der Umgang mit neuen Funktionen anschaulich erklären.

3.5 Die Arbeitsregister des VDC

Dieses Kapitel ist für die unter Ihnen gedacht, die sich nicht mit unserem UTILTIES-Programm begnügen, sondern genau wissen wollen, wie der VDC intern programmiert wird.

Wie Sie ja schon wissen, verfügt der VDC über einen eigenen Speicher. Dieser Speicher kann jedoch nicht, wie beim VIC II, direkt adressiert werden, sondern nur über die Register 18 und 19. Damit ergibt sich nun leider eine doppelt indirekte Ansteuerung, da ja auf die Register auch nur über einen Umweg zugegriffen werden kann.
Beachten Sie bitte, daß sämtliche Register, die irgendeinen Speicherbereich bestimmen, grunsätzlich im HIGH/LOW-Format stehen, also die genau umgekehrt gewohnte Schreibweise.

Um nun die Up-Date-Adresse, das ist der Name dieses Zeigers, zu setzen, muß zunächst die Adresse $D600 mit dem Wert 18 und Adresse $D601 mit dem High-Byte der zu setzenden Adresse geladen werden. Genauso wird dann mit dem Low-Byte verfahren, und nach erfolgter Durchführung zeigt das Up-Date-Registerpaar auf die gewünschte Adresse.

Schön und gut, die Adresse haben wir nun gesetzt, aber wie schreiben wir nun ein Zeichen in das VDC-Ram, oder lesen ein solches aus diesem heraus? - Auch das ist kein Problem, denn es existiert ein weiteres Register, das wir zu Hilfe nehmen können. Es ist das Data-Register mit der Nummer 31. Bei jedem Zugriff auf die Register 18 und 19 wird in dieses DATA-Register der Wert aus dem VDC-Ram geladen, auf den die Zeiger 18/19 zeigen. Sie brauchen, wenn Sie eine bestimmte VDC-Ram-Adresse auslesen wollen, lediglich die Up-Date-Register zu setzen und dann das Register 31 lesen. Wollen Sie einen bestimmten Wert in das VDC-Ram schreiben, so setzen Sie ebenfalls zunächst die Up-Date Register, schreiben aber dann Ihren gewünschten Wert in das Data-Register.

Die letzte Hilfe, die uns der VDC-Chip für die Speichermanipulation noch bietet, ist die folgende:

Das Word-Count-Register, so der Name des Registers, erfüllt grundsätzlich zwei verschiedene Aufgaben. Zum einen ist es möglich, bestimmte Speicherbereiche des VDC-Speichers mit einem bestimmten Wert aufzufüllen. Die andere Aufgabe, die es artig erfüllt, ist das Kopieren ganzer Speicherbereiche innerhalb

des VDC-Speichers einer maximalen Länge von 256 Bytes. Sie werden gleich selber sehen, was es für Vorteile bringt, dieses Register anwenden zu können.
Zunächst wollen wir mit einem einfachen Beispiel anfangen. Lassen Sie uns einfach mit Hilfe des Word-Count-Registers den normalen Bildschrim löschen. Zunächst muß dazu die Up-Date-Adresse, also der Zeiger (Register 18 & 19) auf die Adresse im VDC, die als nächstes bearbeitet werden soll, auf den Begin des Video-Rams gesetzt werden. Danach können wir mit nur 8 Schleifendurchläufen die gesamten 2048 Bytes löschen bzw. diese mit dem Wert des Leerzeichens (32) auffüllen. Ein entsprechendes BASIC-Programm dazu könnte etwa so aussehen:

```
10 rem ***************************************
20 rem *** loeschen des textbildschirms ***
30 rem ***************************************
40 :
50 sys 3113,0,0,32              : rem adresse setzen
60 :
70 for x=1 to 8                 : rem 8 pages
80 :sys 3072,30,0               : =2048 bytes löschen
90 next x
```

Zu dieser kleinen Demonstration muß noch gesagt werden, daß der Wert 0 in diesem Fall dem Wert 256 entspricht. Das ist logisch, denn was für einen Nutzen können wir schon daraus ziehen, wenn wir einen Bereich von null Bytes kopieren oder setzen wollen?

Nun zur zweiten Möglichkeit, dem Kopieren von Blöcken. Zuvor müssen wir durch Setzen des 7. Bits im Register 24 dem Word-Count-Register mitteilen, daß wir ab sofort nur noch Blöcke kopieren und nicht mehr setzen wollen. Wenn Sie nun meinen, daß wir in dem vorhergehenden Fall das Copy-Bit eigentlich zunächst hätten löschen müssen, so haben Sie nur bedingt Recht, denn Sie können sicher sein, daß dieses Bit

gelöscht war, da es bei jedem Zugriff seitens des Betriebsystems gelöscht wird. Das Setzen des Bits erreichen wir nun am besten mit der folgenden BASIC-Zeile:

```
sys 3084,24:rreg a:sys 3072,24,a or 2^7
```

Zuvor muß natürlich unsere kleine BASIC-Erweiterung geladen worden sein.
Wenn sich nun der Computer mit "READY." zurückmeldet, so weiß das Word-Count Register, daß es nun den anderen Aufgabenbereich auszuführen hat.
Aber, wie könnte es anders sein, es folgt sofort das nächste Problem. Wie können wir dem Word-Count-Register mitteilen, welchen Block es wohin kopieren soll? - Nun, auch dieses Problem ist sehr einfach zu lösen. Es gibt zwei Register im VDC, die die Anfangsadresse desjenigen Blocks enthalten müssen, den wir zu kopieren gedenken. Diese Register, es sind die Register 20 und 21, brauchen wir nur mit unserer Adresse zu beschreiben, anschließend das Copybit zu setzen, und durch Beschreiben des Word-Count Registers diesem mitteilen, daß es nun den Block kopieren soll. Auch für diesen zweiten Aufgabenbereich will ich Ihnen ein Beispiel vorstellen, das Ihnen das Prinzip das Scrollings (natürlich in BASIC und deswegen auch nicht all zu schnell) begreiflich machen soll:

```
100 rem **************************************
110 rem ***          scrolling             ***
120 rem ***          ---------             ***
130 rem **************************************
140 :
150 fast                             : rem auf touren bringen
160 :
170 sys 3084,24:rreg a
175 sys 3072,24,aor2^7               : rem copybit setzen
180 :
190 for y=1 to 24                    : rem 24 mal
200 :a0=y*80:a1=(y-1)*80
210 :sys 3072,33,a0-int(a0/256)*256 : rem alten block-
220 :sys 3072,32,int(a0/256)         : rem anfang setzen
```

```
230 :sys 3072,19,a1-int(a1/256)*256 : rem neuen block-
240 :sys 3072,18,int(a1/256)        : rem anfang setzen
250 :sys 3072,30,80                 : rem 80 bytes kopieren
260 next y
270 :
280 sys 3072,24,a                   : rem copybit loeschen
290 sys 3113,a0-int(a0/256)*256,int(a0/256),32
300 sys 3072,30,79                  : rem letze zle. loesch.
```

3.6.1 Der Aufbau des Video-RAM

Wie bereits erwähnt, besitzt der VDC einen eigenen 16K-Speicher (genaugenommen müßte der C-128 deshalb 144K-Computer genannt werden). Dieser Speicher fungiert als Speicher des Zeichensatzes, Farb-RAM und Video-RAM gleichzeitig.

Anzusprechen ist dieser Speicher über die Register 18, 19 und 31 des VDCs. Will man zum Beispiel den Wert 1 in die Speicherstelle 8192 schreiben, so verfährt man wie folgt:

Zuerst wird das High-Byte der Adresse (Reg. 18) durch das im Kapitel über das Utilities-Programm beschriebene Verfahren zur Registeradressierung im Register 18 abgelegt. Ihm folgt in analoger Weise das Low-Byte ins Register 19. Schließlich schreiben wir unseren Wert in Register 31, dem sogenannten Data-Register, das für die Datenaddressierung im VDC-Speicher zuständig ist.
Sie können sich nun selber vorstellen, wie lange diese Prozedur dauerte, würde man dazu BASIC verwenden. Da wir ja bereits über unsere Assemblerroutine verfügen, wollen wir uns nun nicht mehr weiter um Zeitprobleme sorgen, sondern uns deshalb sofort dem eigentlichen Aufbau des VDC-Rams zuwenden.

Es ist normalerweise möglich, auf dem 80-Zeichen-Bildschirm 25 Zeilen zu je 80 Zeichen darzustellen.

Das ergäbe insgesamt eine Kapazität von insgesamt 25*80 = 2000 Zeichen auf dem Monitor und damit ebensoviel Bytes im Speicher.
Weiterhin haben Sie die Möglichkeit, sofern Sie einen Farbmonitor besitzen, diese Zeichen zusätzlich in ihrer Farbe zu variieren. Also werden nicht nur die Zeichen selbst, sondern auch noch ihre farblichen Eigenschaften (Attribute) irgendwo abgelegt. Schließlich sollte irgendwo hinterlegt sein, welche Gestalt die einzelnen Zeichen annehmen sollen.
Alle diese Kriterien finden wir in besagtem 16K-Speicher, der nun wie folgt organisiert ist:

Speicherbereich	Funktion	Länge
$0000-$07CF (0- 1999)	Video-Ram	$07D0 (2000)
$0800-$0FCF (2000- 3999)	Attribut-Ram	$07D0 (2000)
$1000-$1FFF (4096- 8191)	Freier Bereich	$1000 (4096)
$2000-$3FFF (8192-16383)	Zeichengenerator	$2000 (8192)

Diese Speicherbereiche erfüllen im einzelnen nun folgende Funktionen:

In diesem Bereich werden die Zeichen gespeichert, die Sie sehen, wenn z.B. eine Ausgabe mit Hilfe des BASIC-Befehls PRINT tätigen. Jedem Zeichen ist dabei ein anderer Wert zugeordnet, der jedoch nicht mit dem ASCII-Code identisch ist.

Nachschlagen können Sie die den Zeichen entsprechenden Werte entweder im Commodore-128-Bedienungshandbuch auf Seite A-5 oder in unserem Anhang D.

3.6.2 Das Attribute-RAM

Das Attribute-RAM enthält bestimmte Eigenschaften der Zeichen. Durch den Aufbau eines Bytes im Attribute-RAM, wird die Farbe und das Aussehen (revers, unterstrichen oder blinkend) eines Zeichens bestimmt. Dabei hat jedes Bit eines einzelnen Bytes im Attribut-RAM eine bestimmte Funktion:

Bit 7 (ALT): Dieses Bit bestimmt, welcher Zeichensatz des VDC angezeigt werden soll. Der VDC hat soviel RAM "über", daß er leicht in der Lage ist, zwei Zeichensätze zu speichern. Diese Tatsache können Sie feststellen, indem Sie zwar durch die Tastenkombination SHIFT und C= die Zeichensätze umschalten können, die schon dargestellten Zeichen aber, ganz im Gegensatz zum VIC II, sich nicht ebenfalls verändern.

Bit 6 (REV): Ein Setzen dieses Bits bewirkt, daß das Zeichen invertiert dargestellt wird. Von dieser Möglichkeit wird im 128er Betriebsystem kein Gebrauch gemacht, jedoch ist man bei Benutzung dieses Bits in der Lage 512 verschiedene (!!!!) Zeichen und diese noch zusätzlich invertiert darzustellen (insgesamt 1024 Zeichen).

Bit 5 (UNL): Wird dieses Bit gesetzt, so wird das Zeichen an der entsprechenden Stelle unterstrichen. Dadurch könnten in einer Textverarbeitung z. B. das Bild auf dem Bildschirm genau so erscheinen, wie hinterher auf dem Papier. Man würde so maches Blatt Papier einsparen...

Bit 4 (FLASH): Durch Setzen dieses Bits wird das entsprechende Zeichen blinkend dargestellt. Anwendungsmöglichkeit hierfür wäre z. B. das Betonen verschiedener Mitteilungen und Gebote, die auf dem Bildschirm erscheinen sollen.

Bit 3,2,1 und 0: Diese Bits bestimmen die Farbe des Zeichens. Insgesamt können 16 verschiedene Farben dargestellt werden. Auf einem Grün-Monitor könnten Sie dieses durch verschiedene Grau-(vielmehr: Grün-)schattierungen feststellen können.
Hier ist nun eine Tabelle der verschiedenen Bitkombinationsmöglichkeiten:

Bits 3	2	1	0	Farbe
0	0	0	0	Schwarz
0	0	0	1	Dunkelgrau
0	0	1	0	Blau
0	0	1	1	Hellblau
0	1	0	0	Grün
0	1	0	1	Hellgrün
0	1	1	0	Beige-Blau
0	1	1	1	Zynober
1	0	0	0	Rot
1	0	0	1	Hellrot
1	0	1	0	Lila
1	0	1	1	Violett
1	1	0	0	Braun
1	1	0	1	Gelb
1	1	1	0	Hellgrau
1	1	1	1	Weiß

Die Bits 0 bis 3 haben folgende Bedeutung:

3	2	1	0
Rot	Grün	Blau	Intensität

3.6.3 Der Charactergenerator

Der Charakter- oder kurz Char-Generator ist der Teil des Speichers, in dem das Aussehen (nicht die Eigenschaften) eines einzelnen Zeichens festgehalten werden. Sie können jederzeit mit Hilfe unserer Assembler-Routine das Aussehen eines Zeichens verändern. Die Lage einer Zeichendefinition berrechnet sich aus folgender Formel:

Adresse = $2000 + ZN*16

Wobei die Variable ZN die Nummer des Zeichens darstellt.
Ein Beispiel: Um die Adresse des "%"-Zeichens zu ermitteln, schauen Sie im Commodore-Handbuch den entsprechenden Bildschirmcode (nicht ASCII-Code) oder in unserem Anhang nach und ermitteln dann die Adresse des Zeichens:

Adresse = $2000 + 37*16

Adresse = $2250 (8784)

Nun zum Aufbau eines Zeichens: Jedes einzelne Zeichen ist aus 8 Bytes zusammengesetzt. Jedes gesetzte Bit innerhalb eines solchen Bytes stellt auf dem Bildschirm einen gesetzten Punkt dar. Anhand eines weiteren Beispiels soll dies verdeutlicht werden:

7 6 5 4 3 2 1 0	Byte Wert	Adresse
. . * * * * . .	62	$2000
. * * . . * * .	102	$2001
. * * . * * * .	110	$2002
. * * . * * * .	110	$2003
. * *	96	$2004
. * * . . . * .	96	$2005
. . * * * * . .	62	$2006
. . . .	0	$2007

Als Ergänzung zu diesem Beispiel soll hier nun noch ein kleines Programm angefügt werden, das aus dem Prozent-Zeichen einen vollständig ausgefüllten Block macht. So können Sie einfach das Prinzip der Änderung von Zeichen erkennen und in Ihre eigenen Programme aufnehmen.

```
10 rem **********************************
20 rem ***                            ***
30 rem *** aenderung eines zeichens ***
40 rem *** ------------------------- ***
50 rem ***                            ***
60 rem **********************************
70 :
80 wr = dec("c2a")  : rem adresse für 'speicher schreiben'
90 :
100 ad = dec("2000")+ 37 * 16 :rem adresse des '%'-zeichens
110 for z=0 to 8
120 :sys wr,ad-int(ad/256)*256+z,int(ad/256),255
130 next z
```

Wenn Sie dieses Programm starten und dann einige Prozent-Zeichen ausgeben, so werden Sie feststellen, daß sich das Prozent-Zeichen zu einem vollständigen Block verändert hat. Dieses Programm war natürlich nur ein einfaches Beispiel zur Programmierung des Zeichensatzes. Im anschließenden Kapitel werden Sie jedoch ein Programm finden, mit dem die Gestaltung ganzer, eigener Zeichensätze kein Problem mehr darstellen sollte.

3.6.4 Der Grafikspeicher

Der Grafikspeicher bildet in dieser Reihe eine kleine Ausnahme. Da dieser Speicher 16K benötigt, und wir nur 16K zur Verfügung haben, müssen also andere Speicherbereiche ausgeschaltet

bzw. überschrieben werden. Der Speicher beginnt im Normalfall ab der Adresse $0000 (0) und endet bei der Adresse 16000 ($3FCF).

Der Grafikspeicher ist reihenweise aufgebaut, wobei ein gesetzes Bit im Grafikspeicher einem gesetztem Punkt auf dem Bildschirm entspricht. Im Normalfall muß das Attribut-RAM ausgeschaltet werden, da für dieses kein Platz mehr vorhanden ist. Mehr darüber können Sie jedoch im Kapitel über die hochauflösende Grafik erfahren.

3.7 Adreßänderung

Sollten Sie einmal mit der Adreßlage einzelner Speicherbereiche nicht einverstanden sein, so haben Sie die Möglichkeit, diese zu verändern.

Eine Änderung der Startadresse des Video-RAMs geschieht durch Manipulation der Register 12 (High-Byte) und Register 13 (Low-Byte). Diesen brauchen Sie einfach nur Ihre gewünschte Adresse mitzuteilen, und schon befindet sich das Video-RAM an der neuen Position.

Wünschen Sie eine Änderung der Adresse des Attribut-RAMs, so verfahren Sie mit den Registern 20 und 21 analog den Registern 12 und 13.
Damit das Betriebssystem von der neuen Lage der beiden Speicher erfährt, müssen Sie die High-Bytes beider Anfangsadressen den Speicherstellen

$0A2E (2606) Zeiger auf 80 Zeichen Video-RAM
$0A2F (2607) Zeiger auf Attribut-RAM

übergeben. Dabei sollten Sie aber sehr vorsichtig mit der Wahl Ihrer Adreßlagen sein. Beachten Sie immer, daß sich die einzelnen Speicher nicht überschneiden, da es sonst zu sehr seltsamen Effekten kommt.

Ein gutes Anwendungsbeispiel hierzu könnte die Verwaltung mehrerer Textseiten sein. Da wir zwischen Ende des Attribut-RAMs und Anfang des Zeichensatzes noch 4 Kilo-Byte Platz haben, ist dies kein Problem mehr und wurde daher auch in den folgenden Unterroutinen, die mit "GOSUB" angesprungen werden, realisiert.

```
60000 rem ******************************************
60010 rem *** umschalten auf screen 2 ************
60020 rem ******************************************
60030 :
60040 sys 3072,12,int(4096/256)
60050 sys 3072,13,4096-int(4096/256)*256
60060 sys 3072,20,int(6144/256)
60070 sys 3072,21,6144-(6144/256)*256
60080 poke dec("a2e"),int(4096/256)
60090 poke dec("a2f"),int(6144/256)
60100 return
60110 :
61000 rem ******************************************
61010 rem *** umschalten auf screen 1 ************
61020 rem ******************************************
61030 :
61040 sys 3072,12,0
61050 sys 3072,13,0
61060 sys 3072,20,int(2048/256)
61070 sys 3072,21,2048-(2048/256)*256
61080 poke dec("a2e"),0
61090 poke dec("a2f"),int(2048/256)
61100 return
```

Die Adresse des Chargenerators kann, entgegen den Registern 12/13, nur in 8 Kilobyte Schritten verschoben werden. Diese Verschiebung kann mit dem Register 28 erfolgen, bei dem die Bits 5,6 und 7, die Adressbits des Chargenerators 13,14 und 15 darstellen. Da der VDC aber nur über 16 Kilobyte verfügt, ist nur das 5. Bit relevant:

Bit 5	Adresse
0	0000 (0000)
1	2000 (8192)

Der Grafikspeicher kann im Prinzip auch verschoben werden, jedoch ist dies unnütz, da wir ja nur einen einzigen Speicherbereich, in dem die Grafik liegen kann, zur Verfügung haben. Sollte es jedoch dem einen oder anderen Bastler unter Ihnen gelingen, den Speicher des VDC zu erweitern, so sei diesem gesagt, daß der Anfang des Grafikspeichers ebenfalls mit den Registern 12 und 13 festgelegt werden kann.

3.8 VDC Chardesigner

Wie schon angekündigt, wollen wir Ihnen nun ein Programm vorstellen, mit dem sich sehr komfortabel ganze Zeichensätze erstellen lassen. Diese neu erstellten Zeichen können Sie in jedes BASIC-Programm einbauen, da dieser Chargenerator automatisch BASIC-zeilen mit entsprechenden Werten generiert.
Trotz der Länge lohnt sich die Mühe des Abtippens sicher. Ganz Schlaue unter den Lesern wissen bereits jetzt, daß zu diesem Buch eine "Diskette zum Buch" existiert, die einem viel Arbeit erspart und zudem noch Programme präsentiert, die in diesem Buch nicht einmal erklärt sind, und Ihnen bestimmt die Tränen in die Augen treiben werden.

```
100 rem **************************************************
110 rem **                                              **
120 rem **          charaktergenerator vdc              **
130 rem **      --------------------------              **
140 rem **************************************************
150 :
```

```
160 rem *** assembler-programm lesen ***
170 :
180 fast
190 :
200 do while x<59
210 :read a$
220 :a=dec(a$)
230 :poke 3072+x,a
240 :ps=ps+a
250 :x=x+1
260 loop
270 :
280 rem *** assembler-datas ***
290 :
300 data 8d,00,d6,2c,00,d6,10,fb,8e,01,d6,60,8d,00,d6,2c
310 data 00,d6,10,fb,ad,01,d6,a2,00,a0,00,60,48,a9,12,20
320 data 00,0c,68,aa,a9,13,4c,00,0c,20,1c,0c,98,aa,a9,1f
330 data 4c,00,0c,20,1c,0c,a9,1f,4c,0c,0c
340 :
350 graphic 5,1              : rem auf 80 zeichen schalten
360 bank 15
370 for z=0 to 255           : rem 512 zeichen
380 :sys 3113,z,0,z          : rem auf den 80 zeichen
390 :sys 3113,z,1,z          : rem bildschirm
400 :sys 3113,z,9,143        : rem schreiben
410 next z
420 graphic 0,1              : rem auf 40 zeichen zurueckschalten
430 :
440 rem *** variablen vorbereitung ***
450 :
460 e$  = chr$(027) : cl$ = chr$(157) : cr$ = chr$(029)
470 cu$ = chr$(145) : cd$ = chr$(017) : rv$ = chr$(018)
480 nm$ = chr$(146) : ho$ = chr$(019) : cs$ = chr$(147)
490 :
500 xk = 2 : yk = 7 : ch = 0 : zn = 5000
510 :
520 def fn bp(v)= 1024+xk+40*yk
530 :
540 rem *** farbsetzung ***
550 :
```

```
560 color 0,1              : rem rahmenfarbe schwarz
570 color 4,1              : rem hintergrundfarbe schwarz
580 color 5,6              : rem zeichenfarbe gruen
590 :
600 rem *** bildschirmaufbau ***
610 :
620 graphic 0                        : rem auf 40 zeichen schalten
630 :
640 print ho$+ho$;                   : rem evnt. window loeschen
650 print cs$;                       : rem bildschrim loeschen
660 :
670 e1$=chr$(213)                    : rem bogen oben,links
680 for x=1 to 8                     : rem acht
690 :e1$=e1$+chr$(195)               : rem horizontale
700 next                             : rem striche
710 e1$=e1$+chr$(201)                : rem bogen oben,rechts
720 :
730 e2$=chr$(194)                    : rem vertikaler strich
740 for x=1 to 8
750 :e2$=e2$+chr$(32)                : rem 8 * space
760 next x
770 e2$=e2$+chr$(194)                : rem vertikaler strich
780 :
790 e3$=chr$(202)                    : rem bogen unten,links
800 for x=1 to 8                     : rem acht
810 :e3$=e3$+chr$(195)               : rem horizontale
820 next                             : rem striche
830 e3$=e3$+chr$(203)                : rem bogen unten,rechts
840 :
850 print cs$+"          "+rv$+"vdc-chargenerator"
860 for x=1 to 40:print "-";:next
870 :
880 char ,1,6,e1$:char ,14,6,e1$
890 for y=7 to 14
900 :char ,1,y,e2$:char ,14,y,e2$          : rem bildschirm aufbauen
910 next y
920 char ,1,15,e3$:char ,14,15,e3$
930 :
940 char ,25,12,"bildcode :"
950 char ,00,17,"neues zeichen"
```

```
960 char ,14,17,"altes zeichen"
970 :
980 gosub 1530                    : rem altes zeichen darstellen
990 gosub 2170                    : rem bildschrim aktualisieren
1000 :
1010 slow
1020 :
1030 rem ********************************************
1040 rem *** zeichen editieren ***********************
1050 rem ********************************************
1060 :
1070 do
1080 :
1090 zs = peek(fnbp(v))
1100 poke fnbp(v),160
1110 getkey a$
1120 poke fnbp(v),zs
1130 :
1140 rem *** befehlsauswertung ***
1150 :
1160 if a$=cr$ then xk=xk+1           : rem cursorsteuerung
1170 if a$=cl$ then xk=xk-1
1180 if a$=cd$ then yk=yk+1
1190 if a$=cu$ then yk=yk-1
1200 :
1210 if a$=" " then begin             : rem punkt setzen
1220 :poke fnbp(v),81
1230 :xk=xk+1
1240 bend
1250 :
1260 if a$=chr$(160) then begin       : rem punkt loeschen
1270 :poke fnbp(v),32
1280 :xk=xk+1
1290 bend
1300 :
1310 if xk >  9 then xk = 2:yk=yk+1   : rem prueft auf
1320 if xk <  2 then xk = 9:yk=yk-1   : rem bereichsueberschreitung
1330 if yk > 14 then yk = 14
1340 if yk <  7 then yk =  7
1350 :
```

```
1360 if a$=cs$ then begin                : rem feld1 loeschen
1370 :window 2,7,9,14
1380 :print cs$+ho$+ho$
1390 :xk=2 : yk = 7
1400 bend
1410 :
1420 if a$=ho$ then xk=2 : yk = 7       : rem cursor home
1430 if a$="i" then gosub 2650          : rem zeichen invertieren
1440 if a$="w" then gosub 2230          : rem zeichen auswaehlen
1450 if a$="n" then gosub 1760          : rem neues zeichen
1460 if a$="=" then gosub 1970          : rem zeichen uebernehmen
1470 if a$="c" then gosub 2510          : rem altes zchn. = neues zchn.
1480 if a$="^" then gosub 2350          : rem zeichen in datas
uebernehmen
1490 if a$="q" then gosub 2750          : rem quit program
1500 :
1510 loop
1520 :
1530 rem ************************************************
1540 rem *** altes zeichen in feld 2 *****************
1550 rem ************************************************
1560 :
1570 fast
1580 bank 14
1590 z1=7:z2=0
1600 for yk=7 to 14
1610 :for xk=15 to 15+7
1620 ::if (peek(53248+z2+ch*8)and2^z1)=2^z1 then poke
fnbp(v),81:goto1640
1630 ::pokefnbp(v),32
1640 ::z1=z1-1
1650 :next xk
1660 :z1=7:z2=z2+1
1670 next yk
1680 slow
1690 xk=2:yk=7
1700 return
1710 :
1720 rem ************************************************
1730 rem *** neues zeichen in feld 1 *****************
```

```
1740 rem ********************************************
1750 :
1760 fast
1770 bank 15
1780 z1=7:z2=0
1790 for yk=7 to 14
1800 :for xk=2 to 9
1810 ::ad=dec("2000")+16*ch+z2
1820 ::sys 3123,ad-int(ad/256)*256,int(ad/256):rreg a
1830 ::if (a and2^z1) = 2^z1 then poke fnbp(v),81:goto1850
1840 ::poke fnbp(v),32
1850 ::z1=z1-1
1860 :next xk
1870 :z1=7:z2=z2+1
1880 next yk
1890 slow
1900 xk=2:yk=7
1910 return
1920 :
1930 rem ********************************************
1940 rem *** zeichen uebernehmen ********************
1950 rem ********************************************
1960 :
1970 fast
1980 bank 15
1990 z1=7:z2=0:by=0
2000 for yk=7 to 14
2010 :for xk=2 to 9
2020 ::if peek(fnbp(v))=81 then by=by+2^z1
2030 ::z1=z1-1
2040 :next xk
2050 :ad=dec("2000")+16*ch+z2
2060 :sys 3113,ad-int(ad/256)*256,int(ad/256),by
2070 :by=0:z1=7:z2=z2+1
2080 next yk
2090 slow
2100 xk=2:yk=7
2110 return
2120 :
2130 rem ********************************************
```

```
2140 rem *** bildschrim aktualisieren ****************
2150 rem *********************************************
2160 :
2170 char ,35,12,str$(ch)
2180 return
2190 :
2200 rem *********************************************
2210 rem *** zeichen auswaehlen *********************
2220 rem *********************************************
2230 :
2240 char ,0,23:input "zeichencode ? (0..511)";ch
2250 if ch < 0 or ch > 511 then 2240
2260 gosub 1530
2270 gosub 2130
2280 char ,0,23,"                             "
2290 return
2300 :
2310 rem *********************************************
2320 rem *** zeichen in datas abspeichern ************
2330 rem *********************************************
2340 :
2350 print cs$+"sind sie sicher ? (j/n)"          : rem sicherheitsab-
frage
2360 getkey a$
2370 if a$="n" then goto 850
2380 fast
2390 print chr$(147);zn;"data ";                       : rem programm-
zeile
2400 for ad=dec("2000")+16*ch to dec("2000")+16*ch+7   : rem in den
2410 :sys 3123,ad-int(ad/256)*256,int(ad/256):rreg a    : rem tastatur-
2420 :print right$(str$(a),len(str$(a))-1)+",";        : rem buffer
2430 next ad                                           : rem schreiben
2440 sys 3123,ad-int(ad/256)*256,int(ad/256):rreg a      : rem und an-
schliessend
2450 print right$(str$(a),len(str$(a))-1)              : rem zur zeile
2460 print "goto 2480"                                  : rem 2480 sprin-
gen
2470 poke 842,19:poke843,13:poke844,13:poke dec("d0"),3:end
2480 zn=zn+10                                          : rem zeilennum-
mer
```

```
2490 goto 850                                              : rem erhoehen
und
2500 :                                                     : rem bild neu
aufbauen
2510 rem ********************************************
2520 rem *** feld2 nach feld1 kopieren ***************
2530 rem ********************************************
2540 :
2550 bank 15
2560 for yk=7 to 14
2570 :for x=15 to 15+7
2580 ::xk=x:a=peek(fnbp(v))
2590 ::xk=x-13:poke fnbp(v),a
2600 :next x
2610 next yk
2620 xk=2:yk=7
2630 return
2640 :
2650 rem ********************************************
2660 rem *** zeichen invertieren *********************
2670 rem ********************************************
2680 :
2690 for yk=7 to 14
2700 :for xk=2 to 9
2710 ::if peek(fnbp(v))=81 then poke fnbp(v),32:else:poke fnbp(v),81
2720 next xk,yk
2730 xk=2:yk=7:return
2740 :
2750 rem ********************************************
2760 rem *** programm beenden ************************
2770 rem ********************************************
2780 :
2790 char ,0,23,"sind sie sicher ? (j/n)"
2800 getkey a$
2810 if a$="j" then 2830
2820 char ,0,23,"                         ":return
2830 print ho$+ho$+cs$+"delete -5000"
2840 poke 842,19:poke 843,13:poke dec("d0"),2:end
```

Programmerklärung

Zeilen 0 - 260	In diesen Zeilen werden die Datas des Assembler-Programms "Utilities" in den dafür bestimmten Bereich geschrieben.
Zeilen 280 - 330	Diese Zeilen beinhalten das Assembler-Programm in hexadezimaler Form.
Zeilen 350 - 420	Hier wird der gesamte Charaktersatz des VDC auf den 80 Zeichen-Bildschirm geschrieben, um später beim Editieren des Zeichens das neu definierte Zeichen sofort in Orginalgröße betrachten zu können.
Zeilen 440 - 490	Dieser Programmteil bereitet die Variablen vor, die die veschiedenen Steuerzeichen enthalten.
Zeilen 540 - 580	In diesem Programmteil werden die Bildschirmfarben gesetzt.
Zeilen 600 - 830	Diese Zeilen definieren die Variablen, die später den gesamten Bildschirmaufbau enthalten. Wir mußten zu dieser Lösung greifen, auch wenn das Programm dadurch etwas komplexer geworden ist, da das Abtippen diverser Steuerzeichenketten nahezu unzumutbar gewesen wäre.
Zeilen 850 - 960	Dieser Programmabschnitt ist für den eigentlichen Bildschirmaufbau zuständig.
Zeilen 1030 - 1510	Diese Routine beinhaltet die Befehlsauswertung. Von hier werden bei Bedarf die einzelnen Unterroutinen angesprungen und anschließend wieder in diesen Programmabschnitt zurückgekehrt.

Zeilen 1530 - 1700	Diese erste Unterroutine liest ein Zeichen aus dem originalen Zeichengenerator und kopiert es in das 2. Feld.
Zeilen 1720 - 1910	In diesen Zeilen wird ein Zeichen aus dem VDC-Zeichengenerator in das 1. Feld kopiert.
Zeilen 1930 - 2110	Hier wird das erstellte Zeichen aus dem 1. Feld in den Zeichengenerator des VDC übernommen.
Zeilen 2130 - 2180	Diese kurze Unterroutine aktualisiert den Bildschirm.
Zeilen 2220 - 2290	In dieser Routine wird die Nummer des Zeichens, welches Sie zu editieren gedenken, ausgewählt.
Zeilen 2310 - 2490	Hier wird das aktuelle Zeichen in DATA-Statements übernommen, so daß Sie es bei Bedarf in Ihre eigenen BASIC-programme einbauen können.
Zeilen 2510 - 2630	Diese Routine kopiert das Zeichen, das sich im 2. Feld befindet, in das erste Feld.
Zeilen 2650 - 2730	Wird diese Routine angesprungen, so wird das Zeichen im 1. Feld invertiert. D. h. ein Punkt wird gelöscht, wenn er gesetzt ist und umgekehrt.
Zeilen 2750 - 2840	Diese letzte Routine löscht den Chareditor bis auf Ihre erzeugten DATA-Zeilen. Sie können diese dann einfach mit DSAVE abspeichern.

Erklärung des Chargenerators

Nach dem Start des Chargenerators wird zunächst der komplette Zeichensatz des VDC auf den RGB-Monitor geschrieben. Achten Sie bitte darauf, daß die ASCII/DIN-Taste nicht niedergedrückt ist.
Nach einiger Zeit erscheint dann folgendes Bild:

VDC-Chargenerator

Feld 1 Feld 2 Bildcode:

Neues Zeichen Altes Zeichen

In dem 1. Feld können Sie nun Ihr Zeichen erstellen, während im 2. Feld stets das originale Zeichen sichtbar bleibt. Sollte bei einem Befehl kurzfristig das Bild verschwinden, so seien Sie unbesorgt. Nach einigen Momenten wird wieder (fast) alles beim alten sein.
Befehle können Sie dem Programm durch direkten Tastendruck erteilen.

Kommen wir nun zur Erklärung der einzelnen Befehle:

Befehlstaste	Erklärung
'home'	Dieser Tastendruck veranlaßt den Computer, den Cursor in die linke obere Ecke zu plazieren.
'crsr'	Mit den Cursortasten läßt sich der Cursor innerhalb des 8*8-Feldes bewegen. Bereichsüberschreitungen werden dabei automatisch erkannt und abgefangen.
'clr'	Die Taste 'CLR' löscht das 8*8-Feld. ACHTUNG! - Ihr erstelltes Zeichen geht durch diesen Tastendruck sofort verloren. Seien Sie also entsprechend vorsichtig mit dieser Taste!
"C"	Diese Taste bewirkt, daß das Zeichen, welches in dem Feld 'ALTES ZEICHEN' steht, in das Editierfeld kopiert wird. Auch hier würde ein eventuell vorhandenes Zeichen gelöscht.
"I"	Dieser Befehl dient dazu, ein Zeichen in ein reverses umzuwandeln.
"N"	Wenn Sie ein Zeichen, das Sie versehentlich gelöscht haben oder noch einmal editieren möchten, in das Editierfeld holen möchten, so müssen Sie diesen Befehl auswählen.
"W"	Dieser Befehl ist dazu da, ein bestimmtes Zeichen auszuwählen. Der Computer fordert Sie nach diesem Tastendruck auf, eine Zahl zwischen 0 und 511 anzugeben. Wie Sie wissen, ist der VDC-Controler in der Lage, zwei Zeichensätze auf einmal

anzuzeigen. Um an den zweiten Zeichensatz zu gelangen, müssen Sie also 256 zu Ihrem gewünschtes Zeichen addieren, um das eigentliche Zeichen zu erhalten.

"=" Nachdem Sie ein Zeichen fertig erstellt haben, können Sie es durch die "="-Taste in den VDC-Speicher übernehmen. Sie erkennen dann, wie sich das Zeichen auf dem RGB-Bildschirm verändert.

"^" Um nach Beendigung des Programms auch noch etwas von Ihrem gerade erstellten Zeichen zu haben, können Sie es in Datas abspeichern. In Zeile 500 des Charaktergenerators wird mit der Variable ZN die Start-Zeilennummer der DATA-Zeilen angegeben. Sie können diese natürlich entsprechend verändern. Die Zeilennummer darf jedoch keinesfalls den Wert 5001 unterschreiten.

"Q" Wenn Sie all Ihre Zeichen erfolgreich fertiggestellt haben, können Sie das Programm mit "Q"(uit) verlassen. Nach der Sicherheitsabfrage löscht sich das Programm bis auf Ihre erstellten Zeichen selbst. Diese können Sie dann mit dem DSAVE-Befehl auf Diskette speichern oder aber sofort in Ihr Programm übernehmen.

Soweit nun die Befehle des Chargenerators. Ich hoffe, daß sich das Abtippen gelohnt hat und wünsche Ihnen viel Freude bei dem Erstellen Ihrer eigenen Zeichen.

3.9.1 VDC HI-RES-Grafik

Nun ist es soweit. Der viel beschworene, oft verwunschene und doch ersehnte Moment ist gekommen. Wir lüften nun das Geheimnis der hochauflösenden Grafik unseres Commodore 128, von denen die Commodore-Entwickler eine Zeitlang selbst nichts wußten oder nichts wissen wollten. Selbst auf einer der ersten und zweiten Messen, auf denen der Commodore 128 ausgestellt war, konnten die freundlichen Commodore-Standleiter keine oder nur falsche Auskunft über die tatsächliche Grafikfähigkeit ihres eben konstruierten, neu entwickelten, gerade vom Band gelaufenen Rechners geben. Nun, wir wissen es jetzt besser und wollen uns zunächst mit der Speichereinteilung der hochauflösenden Grafik und später mit der eigentlichen Punktberechnung beschäftigen.

Der Grafikspeicher umfaßt 16000 Bytes. Der VDC verfügt aber nur über 16383 Bytes Adressierungs-RAM. Soeben haben wir gelernt, daß sich im VDC-eigenen Arbeitsspeicher neben Video-RAM auch noch das Attribut-RAM und der Zeichensatz tummeln. Logische Schlußfolgerung: Der Zeichensatz muß überschrieben werden. Das mündet natürlich in ein ziemliches Fiasko, denn bei Betätigung der ASCII/DIN-Taste wird unser schönes Grafik-Bild, das wir eventuell später einmal erzeugen können, wieder vernichtet, weil das Betriebsystem das Drücken dieser Taste sofort erkennt und dann den neuen Zeichensatz in die obere Hälfte des VDC-RAMs schreibt, wo natürlich auch Teile unserer Grafik liegen.

Dieses Problem ist auch leider nicht zu lösen, wir müssen halt entsprechend vorsichtiger sein. Doch nun genug der Warnungen, kommen wir zum Aufbau der Grafik:

Die Grafik ist bitweise hintereinander angeordnet. Dabei repräsentiert ein gesetztes Bit im Grafikspeicher einen gesetzten Punkt auf unserem Bildschirm. Die einzelnen Bytes sind nicht wie beim VIC II in 8er Blöcken, sondern hintereinander angeordnet. Das erleichtert und beschleunigt die ohnehin schon sehr

komplizierte Punktberechnung ungemein. Doch nun wollen wir Ihnen zum besseren Verständnis ein kleines Schaubild der hochaufgelösten Grafik geben, mit dem wir dann die Punktberechnung einer einzelnen Koordinate vornehmen können.

Adresse	$0000	$0001	$0002	$0003... - $027F	
$0000	76543210	76543210	76543210	76543210	76543210
$0280	76543210	76543210	76543210	76543210	76543210
$0500	76543210	76543210	76543210	76543210	76543210
$0780	76543210	76543210	76543210	....	
$0A00	76543210				
$0B80	....				
.	7654...				
.	765.				
.	76.				
.	7.				
3B80	....				bis 3E7F

Ziel ist es nun, eine Formel zu entwickeln, die es uns ermöglicht, die Adresse eines Punktes, den wir zu erzeugen gedenken, an eine bestimmte Koordinate zu setzen. Da in einer Reihe 80 Zeichen (sprich: Bytes) stehen und diese wiederum aus 8 Bits bestehen, erhalten wir insgesamt eine horizontale Auflösung von 640 Punkten. In vertikaler Richtung stehen uns genau 200 Bytes zur Verfügung, also erhalten wir in diese Richtung auch eine

Auflösung von 200 Punkten. Um nun die horizontale Position eines Bytes zu bestimmen brauchen wir unsere Koordinate einfach nur ohne Rest durch 8 zu teilen. Wir erhalten also:

reihe = INT(xkord/8)

Um nun die Y-Position nicht zu benachteiligen, müssen wir diese mit 80 multiplizieren und anschließend zu unserer Variable "reihe" addieren. Wir erhalten so unsere endgültige Adresse:

adresse = INT(xkord/8) + ykoord * 80

Wir haben zwar jetzt die gesuchte Adresse, wir wissen jedoch, daß jedes gesetzte Bit einen Punkt darstellt. Da wir nicht 8 Punkte auf einmal setzen möchten, müssen wir nun jetzt noch die Position des Punktes innerhalb des Bytes errechnen. Diese errechnet sich nun aus:

2^(7-(X AND 7)).

Zunächst wird die Bitnummer durch das "AND 7" gebildet. Da jedoch die Bits nicht von vorne nach hinten, also 1, 2, 3, 4, 5 usw., sondern in genau umgekehrter Reihenfolge angeordnet sind (also 7, 6, 5, 4 usw.), müssen wir das Byte durch das 7-(..) umdrehen. Das Ganze wird dann noch zur Basis 2 potenziert, um den eigentliche Bit-Wert zu erhalten. Unsere vollständige Formel lautet dann:

ADD = INT(XK/8) + YK*80

BYT = 2^(7-(XK and 7))

Der Bytewert, den wir dabei erhalten, darf jedoch nicht einfach an diese Adresse geschrieben werden, sondern muß quasi als Maske mit ihm logisch verknüpft werden. Dazu noch eine Tabelle, die Ihnen das Setzen und Löschen von Punkten vereinfachen soll:

Punkt Setzen : BYT = PEEK(AD) OR BYT
Punkt Löschen : BYT = PEEK(AD) AND NOT BYT
Punkt Invertieren : BYT = EOR (PEEK(AD) , 128)

Nun haben wir das schwierigste überstanden. Jetzt fehlt nur noch das Einschalten und Löschen der Grafik, wovon Sie, wie ich annehme, angenehm überascht sein dürften. Wir verwenden zum weiteren Arbeiten wieder unsere Utilities, die ja schon bei der Zeichensatzprogrammierung eine große Hilfe waren.

Wenn wir in die Registerbeschreibung des VDC schauen, so finden wir in der Beschreibung des Registers 25 den Hinweis, daß das Setzen des 7. Bits ein Einschalten der Grafik und das Löschen des 6. Bits ein Ausschalten des Attribut-RAMs, dessen Speicher wir ja nun nirgends mehr unterbringen können, zur Folge hat. Wir nehmen also den Standardwert 71 und addieren 2^7, um das 7. Bit zu setzen, und subtrahieren 64, damit das 6. Bit gelöscht wird. Dieses Ergebnis schreiben wir schließlich wieder in Register 25:

SYS 3072,25,71+128-64

Nun müßte die Grafik eingeschaltet sein, aber auf dem Bildschirm noch ein ziemliches Wirrwarr erscheinen. Sie können nun in der unteren Bildschirmhälfte den Zeichensatz und die Vorgehensweise des Betriebssystems erkennen, wenn die ASCII/DIN-Taste gedrückt wird. Doch nun weiter im Text: Wir müssen unsere Grafik noch löschen. Auf den ersten Blick erscheint es Ihnen bestimmt am einfachsten, jedes Byte des VDC-Speichers einzeln zu löschen, doch erinnern wir uns an die Speicherverwaltungsregister, die wir ja schon kennengelernt haben.
Wir benutzen das Word-Count-Register, um jeweils 256 Bytes des Grafikspeichers auf einmal zu löschen. Dadurch erreichen wir selbst in BASIC einen solchen Geschwindigkeitsvorteil, daß dieser selbst auf dem herkömmlichen Wege in Maschinensprache nicht mehr übertroffen werden kann.

Hier sei nun das Programm aufgelistet, das die Aufgabe hat, die Grafik einzuschalten und zu löschen:

```
10 sys 3072,25,135          : rem grafik einschalten
20 :30 sys 3113,0,0,0       : rem up-date adresse und Data-
                              register setzen
40 :50 for x=0 to 63        : rem 63*256=16128 Bytes löschen
60 : sys 3072,30,0
70 next x
```

Wenn wir dieses Programm jetzt auch noch im 2 Megaherz-Modus laufen lassen, so erhalten wir für BASIC eine kaum zu glaubende Geschwindigkeit.

Um unsere Grafik auch noch in der Farbe variieren zu können, dient uns nun Register 26, dessen Nibbles jeweils für Hintergrund oder Punktfarbe zuständig sind.
Wir ergänzen also unser Programm:

```
80 :90 sys 3072,26,8*16+0
```

Und erhalten nun, wenn wir einen oder mehrere Punkte setzen, diese in einer schönen, leuchtenden Farbe.

Das nun folgende Programm zeichnet Ihnen nun eine Grafik auf den RGB-Monitor, die Sie bestimmt eine ganze Weile bestaunen werden:

```
100 rem *********************************************
110 rem ***                                       ***
120 rem ***   sinus/cosinus kurve (640*200)       ***
130 rem ***   -----------------------------       ***
140 rem ***                                       ***
150 rem *********************************************
160 :
170 rem grafik einschalten
180 :
190 sys 3072,25,128+7        : rem grafik ein, attrbt. aus
200 sys 3072,26,9*16         : rem farbe = rot
210 :
220 rem grafik loeschen
```

```
230 :
240 sys 3084,24:rreg a        : rem copy-bit loeschen
250 sys 3072,24,a and not 2^7
260 :
270 sys 3113,0,0,0            : rem adresse setzen
280 :
290 for x=1 to 63             : rem 64*256 bytes loeschen
300 :sys 3072,30,255
310 next x
320 :
330 y=99                      : rem x-achse zeichnen
340 for x=0 to 639
350 :gosub 530
360 next x
370 :
380 x=319                     : rem y achse zeichnen
390 for y=0 to 199
400 :gosub 530
410 next y
420 :
430 :
440 for x=0 to 639 step .5
450 :y=int(40*sin(x/25)+100) : rem sinus-wert berechnen
460 :gosub 530                : rem punkt plotten
470 :y=int(40*cos(x/25)+100) : rem cosinus-wert berechnen
480 :gosub 530                : rem punkt plotten
490 next x
500 :
510 end
520 :
530 rem **************************
540 rem *** punkt berechnung *****
550 rem **************************
560 :
570 ad=int(x/8)+y*80          : rem adr. eines 8ter blocks
580 by=2^(7-(x and 7))        : rem adr. in einem 8ter block
590 lb=ad-int(ad/256)*256     : rem low-byte errechnen

610 :
620 sys 3123,lb,hb:rreg a     : rem adresse auslesen
```

```
630 by = a or by              : rem mit oder verknuepfen
640 sys 3113,lb,hb,by         : rem und abspeichern
650 return
```

Damit sind wir jedoch noch lange nicht am Ende der Grafikmöglichkeiten des VDC. Durch einen kleinen Trick ist es nämlich möglich, wenn auch mit Einschränkungen, Grafiken mit mehr als einer Farbe zu erzeugen!!!

3.9.2 VDC HI-RES-Multi-Color-Grafik

Ja, Sie haben wirklich richtig gelesen. Es ist tatsächlich möglich, eine farbige Multi-Color-Grafik auf den RGB-Bildschim zu zaubern.
Wir hatten ja schon festgestellt, daß man aus Speicherplatzmangel keine Verwaltungsmöglichkeit für das Attribute-RAM hat, das seinen Dienst ebenso gut im Hires-Modus ausführt. Diesen Platz beschaffen wir uns, indem wir einfach vom Grafikspeicher ein paar Bytes "stehlen". Das wirkt sich zwar etwas ungünstig auf die Auflösung aus, denn es steht uns dann "nur" noch eine Auflösung von 640*180 Pixels zur Verfügung, aber das sollte für die notwendigsten Anwendungsbereiche ausreichen.

Bevor wir nun der Ironie verfallen, lassen Sie uns zunächst mit den theoretischen Grundlagen beginnen.

Zunächst müssen wir bestimmen, wo unser neues Farb-RAM liegen soll. Da der VDC sowieso nur 16000 Bytes für die Grafik braucht, brauchen wir dem Grafikspeicher nur 1665 Bytes zu entziehen. Unser neuer Attribut-RAM-Start liegt damit an der Adresse 14336. Wir können das dem VDC durch die Register 20 und 21 mitteilen.
Jetzt müssen wir eine Methode zur Farbberechnung finden. Dazu errechnen wir zuerst den X-Koordinatenanteil der Adresse. Wir müssen ganz einfach die X-Koordinate durch 8 teilen, da ja in einer Zeile 80 Bytes sind und die Auflösung pro Zeile 640 Pixels beträgt (640/80 = 8).

Nun müssen wir noch den Y-Anteil der Adresse berechnen. Wir teilen hierzu die Y-Koordinate ebenfalls durch 8, weil ein 8*8 Farbblock, wie der Name schon sagt, 8 horizontale Anteile besitzt. Da aber eine Zeile aus 80 Bytes besteht, müssen wir dieses Ergebnis noch mit 80 multiplizieren.

Als letztes muß nun noch die Basisadresse des Attribut-RAMs hinzuadiert werden, um die endgültige Adresse zu erhalten. Wir erhalten also folgende Formel:

ad = 14336 + int(x/8) + (80 * int(y/8))

Das Byte, das nun an diese Stelle geschrieben wird, entspricht im Aufbau demjenigen, das im Kapitel über den Aufbau des Attribut-RAMs geschrieben steht.

Hier ist nun zum Schluß noch ein kleines Demo-Programm, daß Ihnen die Farbmöglichkeiten in der hochaufgelösten -Grafik einmal demonstrieren soll:

```
100 rem **********************************************
110 rem ***                                        ***
120 rem ***   sinus/cosinus kurve (640*200)        ***
130 rem ***       (!!! mehrfarbig !!!)             ***
140 rem ***   ----------------------------         ***
150 rem ***                                        ***
160 rem **********************************************
170 :
180 rem grafik einschalten
190 :
200 sys 3072,25,128+64+7      : rem grafik ein, attribut ein
210 sys 3113,0,56,0           : rem atrb. adr. setzen
220 :
230 rem atribut-ram nach 14334 legen
240 :
250 sys 3072,20,56            : rem low-byte setzen
260 sys 3072,21,0             : rem high-byte setzen
270 :
280 rem grafik & atribut-ram loeschen
```

```
290 :
300 sys 3084,24:rreg a        : rem copy-bit loeschen
310 sys 3072,24,a and not 2^7
320 :
330 sys 3113,0,0,0            : rem adresse setzen
340 :
350 for x=1 to 64             : rem 16384 bytes loeschen
360 :sys 3072,30,0
370 next x
380 :
390 for x=0 to 619 step 2
400 :y=int(20*sin(x/25)+50)  : rem sinus-wert berechnen
410 :c=2:gosub 500            : rem punkt und farbe
420 :y=int(20*sin(x/25)+100) : rem setzen
430 :c=4:gosub 500
440 :y=int(20*sin(x/25)+150)
450 :c=6:gosub 500
460 next x
470 :
480 end
490 :
500 rem ***************************
510 rem *** punkt berechnung *****
520 rem ***************************
530 :
540 ad=int(x/8)+y*80          : rem adresse eines 8ter blocks
550 by=2^(7-(x and 7))        : rem adresse in einem 8ter block
560 lb=ad-int(ad/256)*256     : rem low-byte errechnen
570 hb=int(ad/256)            : rem high-byte errechnen
580 :
590 sys 3123,lb,hb:rreg a     : rem adresse auslesen
600 by = a or by              : rem mit bytewert oder verknuepfen
610 sys 3113,lb,hb,by         : rem und bytewert abspeichern
620 :
630 rem ***************************
640 rem *** farbe berechnen ******
650 rem ***************************
660 :
670 ad=14336+int(x/8)+(int(y/8)*80)
680 by=c                      : rem farbe = bytenr.
```

```
690 lb=ad-int(ad/256)*256     : rem low-byte errechnen
700 hb=int(ad/256)            : rem high-byte errechnen
710 :
720 sys 3113,lb,hb,by         : rem und bytewert abspeichern
730 :
740 return
```

3.10.1 Bildschirmformat Änderung

Auch diese Überschrift läßt nur ahnen, was sich hinter ihr versteckt. Doch mit etwas Optimismus kann man diese Überschrift doch korrekt interpretieren. Sie werden es kaum glauben, aber die VDC kann mehr als 25 Zeilen zu je 80 Spalten verwalten. Stellen Sie sich doch einmal vor, wie komfortabel eine Datenverwaltung mit einem größeren Bildschirmformat wäre. Man könnte den gesamten Bildschirmbereich für eine Maske benutzen und müßte nicht, wie es oft bei Textverarbeitungen für den Commodore 64 der Fall war, eine oder gar zwei Zeilen für die Statuszeile, auf die man nun einmal nicht verzichten kann, opfern. Doch bevor wir nun ganz der Träumerei verfallen, lassen Sie uns sie lieber verwirklichen:

Sie werden in der VDC-Registerbeschreibung bestimmt schon diverse Erklärungen gelesen haben, mit denen Sie nichts anfangen konnten. Dazu gehörte garantiert auch das Register 6. Dieses Register bestimmt nämlich die Anzahl der Zeilen, die auf dem Bildschirm erscheinen. Wenn Sie dieses Register inkrementieren, so werden Sie feststellen, daß zu unserem normalen Bildschirmfenster drei weitere Zeilen hinzugekommen sind. Diese Zeilen können jedoch mit dem normalen Bildschirmeditor nicht mehr angesprungen werden.

Behelfen wir uns deswegen mit einer kleinen BASIC-Unterroutine, die eine in der Variablen "ED$" gespeicherte Zeichenkette an einer beliebigen Position ausdruckt.

```
60000 rem ***********************************
60010 rem ***                             ***
60020 rem *** letzten zeilen einschalten ***
60030 rem *** ------------------------- ***
60040 rem ***                             ***
60050 rem ***********************************
60060 :
60070 sys 3072,6,28       : rem 28 zeilen einschalten
60080 sys 3072,7,34       : rem bildschirm neu syncr.
60090 na = dec("1000")    : rem neue attributramlage
60100 lb = na and 255     : rem low-byte errechnen
60110 hb = int(na/256)    : rem high-byte errechnen
60120 sys 3072,20,hb      : rem high-byte setzen
60130 sys 3072,21,lb      : rem low-byte setzen
60140 poke 2607,hb        : rem a.e. bekanntgeben
60150 return
60160 :
60170 rem *********************************
60180 rem ***                           ***
60190 rem *** letzten zeilen loeschen   ***
60200 rem *** ---------------------     ***
60210 rem ***                           ***
60220 rem *********************************
60230 :
60240 c=32:ad=2000        : rem videoram loeschen
60250 gosub 60280
60260 c=01:ad=4096        : rem attributram loeschen
60270 :
60280 lb=ad and 255       : rem low-byte ermitteln
60290 hb=int(ad/256)      : rem high-byte ermitteln
60300 sys 3113,lb,hb,c    : rem diese adresse setzen
60310 sys 3072,30,240     : rem und die naechsten
60320 return              : rem 240 bytes loeschen
60330 :
60340 rem *********************************
60350 rem ***                           ***
60360 rem ***   string "ed$" an x,y     ***
60370 rem ***   -------------------     ***
60380 rem ***                           ***
60390 rem *********************************
```

```
60400 :
60410 l=len(ed$)           : rem laenge des strings
60420 ad=x+80*y            : rem adresse errechnen
60430 lb=ad and 255        : rem low-byte errechnen
60440 hb=int(ad/256)       : rem high-byte errechnen
60450 for x=1 to len(ed$)
60460 :by=asc(mid$(ed$,x,1)) and dec("3f")
60470 :sys 3113,lb,hb,by
60480 :lb=lb+1:if lb>255 then lb=0:hb=hb+1
60490 next x
```

Nun zur Anwendung dieser drei Unterroutinen.

Mit der ersten läßt sich der Bildschirm zunächst auf das neue Format bringen. Gleichzeitig wird dort die neue Synchronisation des Bildschirms eingestellt, um unsere neuen Zeilen auch sichtbar zu machen.
Die zweite Unterroutine hat die Aufgabe, den Inhalt der neuen Zeilen, in denen ja nur ein großes Zeichenwirrwar herrscht, zu löschen.
Die letzte Unterroutine ist auch schließlich die wichtigste. Sie positioniert den String "ED$" an den Koordinaten X und Y, die in Variablen übergeben werden müssen. Wir raten Ihnen, diese Routinen im 2-MHz-Modus zu betreiben, da sie sonst wegen der relativ langen Ausführungszeit, speziell wegen des Schreibens der Stringvariable auf den Bildschirm, zu langwidrig wäre.

3.10.2 Alles über den Cursor

Sie werden bestimmt schon festgestellt haben, daß der VDC den Cursor selbständig blinken läßt, und dieser auch dem entsprechend einfach manipuliert werden kann. In der Tat, gibt es zahlreiche Register, die für diesen Cursor zuständig sind.

Zunächst kann der VDC-Cursor durch die Register 14 und 15 an die bestimmte Position gesetzt werden, die durch die Formel

curadr = basisvr + x + 80*y

errechnet werden, wobei "curadr" die Cursoradresse und "basisvr" die Basisadresse des Video-RAMs darstellen.

Aber das ist lange noch nicht alles, was man mit dem VDC-Cursor anstellen kann. Sie werden bestimmt schon gemerkt haben, daß man durch die Zeichenfolge "ESC + 'U'" den Cursor als kleinen Strich darstellen kann, der sich "unter" den Zeichen herbewegt.

Es ist nun so, daß man die Rasterzeile, in der der Cursor startet und endet, mit den Registern 11 und 12 ändern kann. Dabei muß jedoch bei der Manipulation des Registers 11 beachtet werden, daß der neue Wert, den Sie in dieses Register zu schreiben gedenken, grundsätzlich mit einer logischen Verknüpfung des vorherigen Inhalts geschieht. Sie sollten also zunächst das Register 11 auslesen, die unteren Bits mittels "AND NOT bitmuster" eliminieren und anschließend eine "OR"-Verknüpfung mit Ihrem neuen Wert machen.

Das gerade Gesagte sollten Sie übrigens bei allen VDC- und auch VIC II-Register beherzigen, bei denen ein einzelnes Register für mehrere Aufgaben zuständig ist.

Mit eben diesem Register kann nun auch die Frequenz, mit der der Cursor blinkt geändert werden. Diese Frequenz kann in insgesamt vier Schritten von "Cursor blinkt nicht" über "Cursor steht" und "Cursor blinkt mit normal" bishin zum schnellen Blinken des Cursors eingestellt werden.

3.10.3 Das Blinken und Unterstreichen von Zeichen

Ebenso wie die Blinkfrequenz des Cursors eingestellt werden kann, ist die Geschwindigkeit des Zeichenblinkens zu manipulieren. Das 5. Bit im Register 24 ist für dieses Blinken verantwortlich. Ist es gelöscht, so blinken alle Zeichen mit 1/16 der Bildwiederholfrequenz, also normal. Bei gesetztem Bit jedoch blinken die Zeichen mit 1/32 der Bildwiederholfrequenz, also sehr viel schneller. Für Anwenderprogramme kann dies sehr von Vorteil sein, z. B. wenn das Programm dem User mitteilen möchte, in welchem Modus er sich gerade befindet. Oder auch um die Dringlichkeit eines aufgetretenen Fehlers zu betonen, ist dieses Blinken sehr von Nutzen.

Auch das Unterstreichen von Zeichenketten kann für die obigen Anwendungsbereiche sehr wichtig sein. Der VDC gibt uns jedoch mit dem Register 29 die Möglichkeit, aus dem Unterstreichen ein Durch- oder Überstreichen zu machen. Auch das kann seinen Zweck in so manchen Anwendungsbereichen haben. Man denke nur wieder an eine Textverarbeitung...

IV. Grundlagen der Grafikprogrammierung

4.1 Linie

Bisher haben wir nur Befehle kennengelernt, die uns die Verwaltung der HI-RES-Grafik so gut wie abgenommen haben. Doch es wird bestimmt einige unter Ihnen geben, denen es nicht reicht, dem Computer zu sagen "DRAW 1,0,0 to 319,199", sondern die genaustens wissen wollen, wie der Computer die einzelnen Punkte berechnet, die auf dieser Linie liegen. Für diese Leser ist das folgenden Kapitel gedacht, das sich ausschließlich mit den Grundlagen der Grafikprogrammierung beschäftigt.

Schon etwas schwieriger gestaltet sich das Zeichnen einer Linie zwischen zwei beliebigen Punkten auf dem Bildschirm. Man sieht dies zwar täglich in irgendwelchen Programmdemos, macht sich jedoch nie richtig Gedanken darüber, welche Überlegungen dahinterstecken. Das Problem ist: Wie stelle ich fest, welche Punkte des Bildschirms auf dieser Linie liegen. Um es zu lösen, müssen wir uns ein wenig mit der sogenannten Analytischen Geometrie beschäftigen. Bekommen Sie keinen Schreck! Hinter diesem monströsen Begriff verbirgt sich etwas ganz Harmloses (jedenfalls in dem Rahmen, der uns hier interessiert), und wenn es Sie nicht so sehr interessieren sollte, etwa weil sich Ihnen damit üble Kindheitserinnerungen verbinden, dann können Sie die folgenden Zeilen ruhig überlesen. Was wir suchen, ist eine Formel, mit der wir die Punkte einer Geraden berechnen können, deren Endpunkte gegeben sind.

Nahezu jeder von uns wird schon einmal in irgendeinem Zusammenhang (meist aus der Schule her) von der sogenannten nominierten Geradengleichung gehört haben:

$$y = mx + n$$

Wobei x und y die Koordinaten eines Punktes auf einer Geraden, m die Steigung der Geraden und n der Schnittpunkt mit der Y-Achse darstellt. Durch einfache Umformung dieser Geraden erhalten wir:

$n=y-mx$

Kennen wir nun zwei Punkte der Geraden (unsere Endpunkte x1,y1 und x2,y2), so können wir gleichfalls die zwei folgenden Formeln aufsetzen, die wir gleichsetzen können:

$$n = y1 - mx1 \quad ---- \quad n=y2-mx2$$

$$y1 - mx1 = y2 - mx2$$

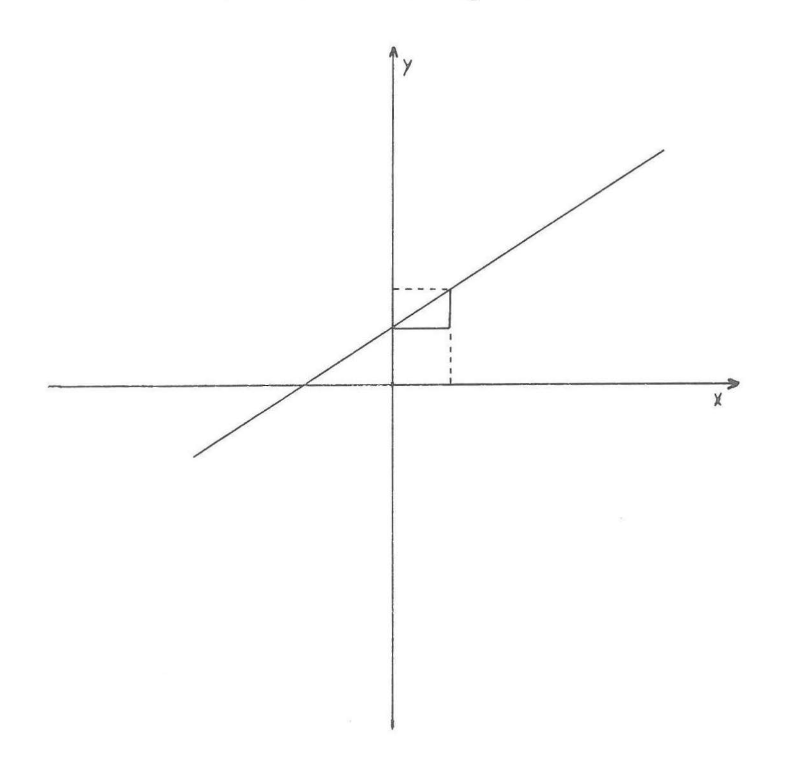
$$\Leftrightarrow \qquad m = \frac{y2-y1}{x2-x1}$$

Letztere Formel läßt uns nun die Steigung m der obigen Gleichung ausrechnen. Ist n = 0, so geht die Gerade durch den Ursprung mit den Koordinate 0,0. Verschieben wir diesen Ursprung der Geraden zu einem Endpunkt, so müssen wir entsprechend die beiden Koordinaten (in diesem Fall x2 und y2) zu X und Y addieren. Die folgende Formel gibt uns nun die endgültige Geradengleichung wieder, die bereits die verschobene Gerade angibt und in die m eingesetzt wurde:

$$y = \frac{y2-y1}{x2-x1} * (x - x2) + y2$$

Diese Formel ist die Grundlage des unten dargestellten Programms und wird stückweise in den Zeilen 380, 480 und 500 errechnet, wobei die x-Koordinate XK stets von x2 nach

x1 läuft, und für jeden solchen X-Wert der entsprechende Y-Wert bestimmt wird. Ein Schaubild mag diese Formel erläutern:



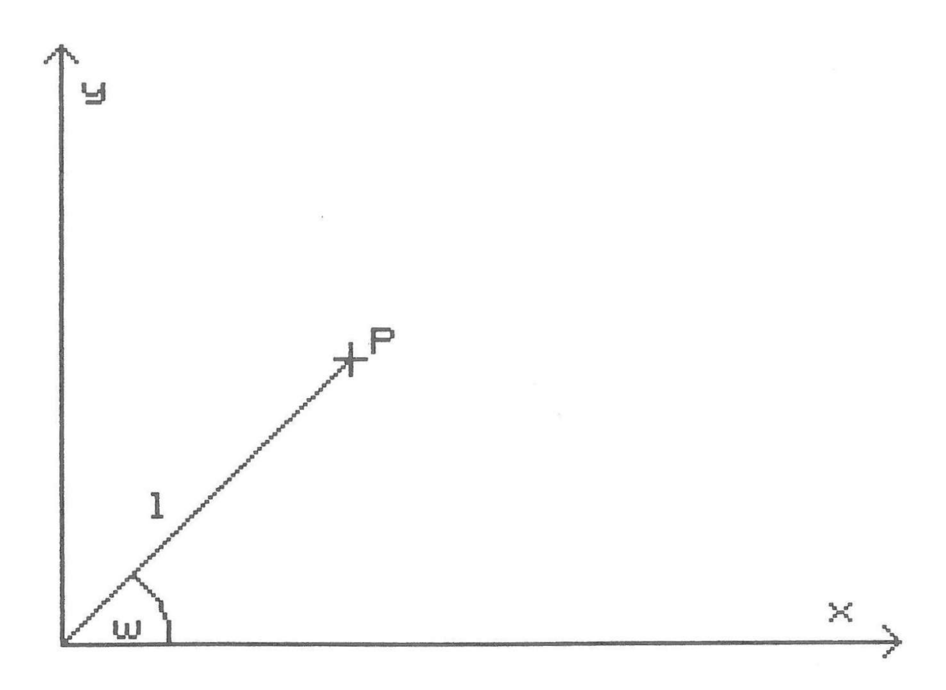
Das einzige Problem bei dieser Formel entsteht, wenn wir eine Senkrechte zeichnen wollen. In diesem Fall wird x1=x2 und damit der Nenner der Steigung gleich 0, was zu einem "DIVISION BY ZERO ERROR" führt. Wir umgehen diese Unkorrektheit, indem wir mit den zwischen den Befehlen "BEGIN" und "BEND" stehenden Argumenten eine Senkrechte zeichnen. Wie Sie selbst sehen werden kommt ein BASIC-programm in der Geschwindigkeit mit einem Assemblerprogramm natürlich nicht mit. Trotzdem kann diese Routine Ihnen helfen, die Funktionsweise des Berechnens von Linien zu verstehen.

```
100 rem ******************************
110 rem ***                        ***
120 rem ***  zeichnen einer linie  ***
130 rem ***  --------------------  ***
140 rem ***                        ***
150 rem ******************************
160 :
170 color 4,1                 : rem rahmenfarbe schwarz
180 color 0,1                 : rem vordergrundfarbe schwarz
190 color 1,6                 : rem zeichenfarbe gruen
200 :
210 graphic 1,1               : rem grafik einschalten & loeschen
220 x1=110:y1=120             : rem startkoordinaten festlegen
230 x2=130:y2=140             : rem endkoordinaten festlegen
240 :
250 gosub 380                 : rem gerade zeichnen
260 :
270 getkey a$                 : rem auf taste warten
280 graphic 0                 : rem graphic ausschalten
290 list                      : rem programm listen
300 end                       : rem programm beenden
310 :
320 rem ***************************
330 rem ***                     ***
340 rem ***   linie zeichnen    ***
350 rem ***                     ***
360 rem ***************************
370 :
380 dy=y2-y1 : dx=x2-x1       : rem differenzen errechnen
390 yk=y2 : xk=x2             : rem y-start
400 :
410 if dx=0 then begin        : rem senkrechte linie zeichnen
420 :for yk=y2 to y1 step sgn(-dy)
430 ::gosub 560               : rem punkt zeichnen
440 :next yk
450 :return                   : rem fertig
460 bend
470 :
480 dd=dy/dx                  : rem steigung
490 for xk=x2 to x1 step sgn(-dx)
```

```
500 : zk=int(dd*(xk-x2)+y2)          : rem geradengleichung
510 : do while zk<>yk
512 ::   yk=yk+sgn(-dy)
514 ::   gosub 560
516 : loop
520 : gosub 560
530 next xk                          : rem naechste x-koordinate
540 return
550 :
560 draw ,xk,yk : xk=xk+1
570 draw ,xk,yk : xk=xk-1            : rem doppelt breit zeichnen
580 return
```

4.2 Das Polarkoordinatensystem

An dieser Stelle wollen wir auf eine alternative Form der Koordinatenfestlegung in einem Koordinatensystem eingehen. Es ist das sogenannte Polarkoordinatensystem, auf das auch noch in den Statistikprogrammen verwiesen wird. Doch was zeichnet nun das Polarkoordinatensystem aus? Im Gegensatz zum normalen Koordinatensystem, in dem ja ein Punkt durch eine X- und eine Y-Koordinate festgelegt ist, ist ein Punkt im Polarkoordinatensystem durch die Länge der Geraden zwischen dem Punkt P und dem Koordinatenursprung sowie dem Winkel zwischen dieser Geraden und dem positiven Teil der X-Achse definiert. Dies mag im ersten Moment relativ kompliziert klingen, doch es ist relativ einfach es anhand der folgenden Zeichnung zu verdeutlichen:



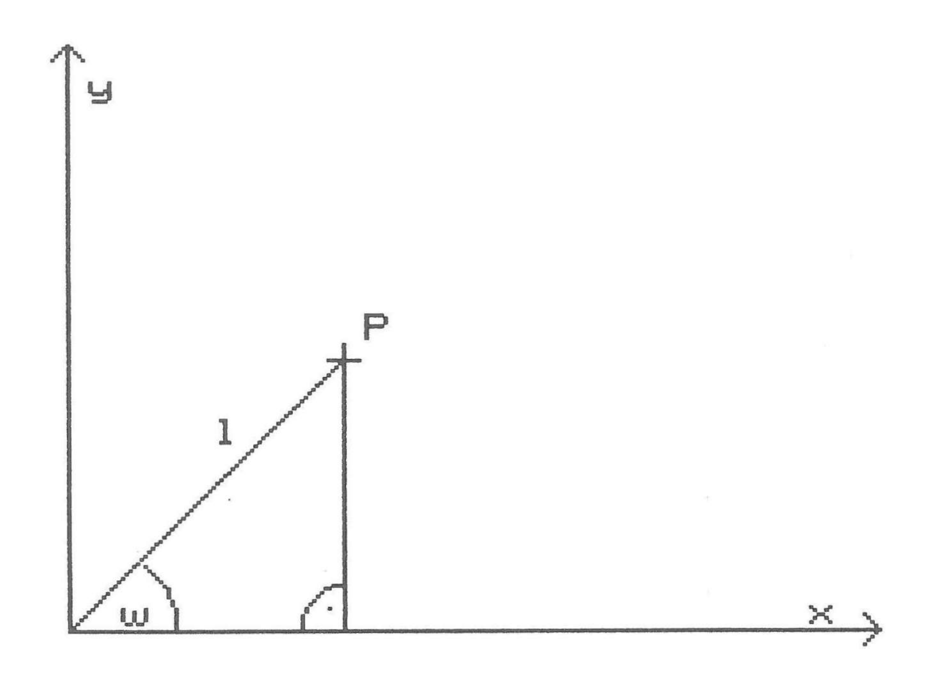
Doch was haben wir nun von dieser, im Gegensatz zum normalen Koordinatensystem relativ komplizierten, Bestimmung eines Punktes? Nun, es sprechen mehrere Grunde dafür:

1. Es lassen sich sehr Kreise um einen Punkt ziehen, indem einfach der Winkel W geändert, und die Entfernung zum Koordinatenursprung L beibehalten wird.
2. Es können auch Kreisausschnitte bezeichnet werden, oder einzelne Punkte eines Kreises an einem bestimmten Winkel berechnet werden. Dies ist z.B. nötig, wenn man den Radius eines Kreises oder einer Ellipse einzeichnen will.

Obwohl diese Punkte alle für die Verwendung eines Polarkoordinatensystems sprechen, gibt es noch einen Punkt, der uns die

Anwendung des Polarkoordinatensystems etwas erschwert. Es ist die Umrechnung dieser Koordinaten in die normalen X- und Y-Angaben, denn der Computer versteht ja nur diese Koordinaten. Um die Koordinaten umzurechnen, müssen wir uns etwas mehr mit den Winkelsätzen im rechtwinkligen Dreieck beschäftigen. Sollten Sie in diesem Fachbereich der Mathematik keine Leuchte sein, so können Sie die folgende mathematische Erklärung ruhig überlesen. Für Sie sind dann eigentlich nur die beiden Formeln zur Umrechnung wichtig. Doch hier jetzt die Erklärung der mathematischen Vorgehensweise:

Als erstes fällen wir ein Lot vom Punkt P zur X-Achse. Wir erhalten dann ein rechtwinkliges Dreieck, das aus einem Teil der X-Achse (Ankathete), einem Teil der Parallelen zur Y-Achse (Gegenkathete) und der Strecke L (Hypothenuse) gebildet wird. Unser Polarkoordinatensystem sieht dann jetzt so aus:



Bekannt sind uns die Strecke L und der Winkel W. Mit diesen beiden Angaben können wir nun die Kathete und die Gegenkathete des Dreiecks berechnen:
Für die Berechnung der Gegenkathete, also des Y-Wertes, gilt:

Y = L * SIN (W)

während für die Berechnung des X-Wertes folgende Formel gilt:

X = L * COS (W)

Wollen Sie nun eine Ellipse zeichnen, so können Sie in beiden Formeln auch verschiedene Werte für L einsetzen. Das könnte dann beispielsweise so aussehen:

Y = YR * SIN (W) und X = XR * COS (W)

wobei außer den genannten Parametern bedeuten:

XR : Radius der Ellipse in X-Richtung

YR : Radius der Ellipse in Y-Richtung

Nun kommt es aber selten vor, daß wir einen Kreis oder eine Ellipse direkt um den Koordinatenursprung zeichnen wollen. Aus diesem Grund verschieben wir unseren Koordinatenursprung zum gewünschten Kreismittelpunkt. Die endgültigen Formeln sehen dann so aus:

X = MX + XR * COS (W) und Y = MY + YR * SIN (W)

wobei MX und MY jetzt die Koordinaten des Kreismittelpunktes darstellen.
Da der Winkel W in Radiant angegeben werden muß, wir aber in Grad rechnen hier noch die Formeln zur Umrechnung von Grad in Radiant und umgekehrt:

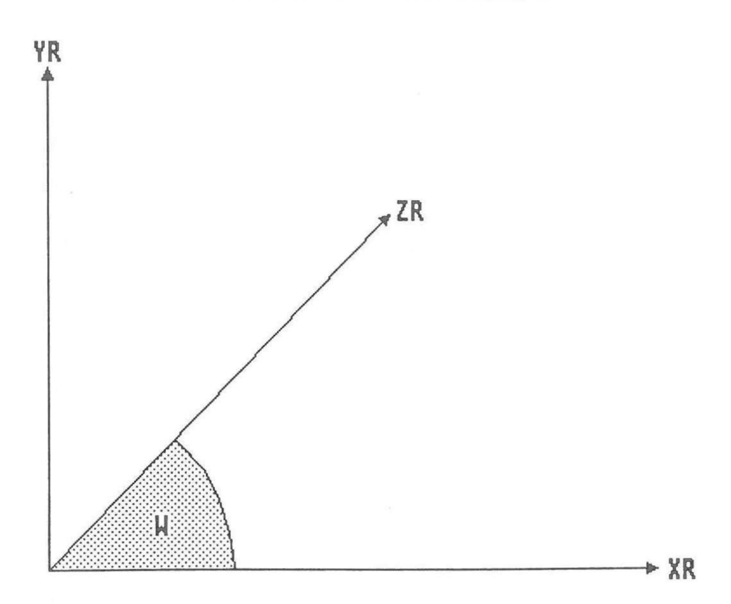
Radiant = Grad * (3.1415 / 180)
Grad = Radiant / (3.1415 / 180)

Probieren Sie doch nun einmal das oben erlangte Wissen selbständig in einem kleinen Programm aus! Setzen Sie z.B. den X-Radius auf 50, den Y-Radius auf 40 und erhöhen Sie in einer Schleife den Winkel W. Nachdem Sie dann die Koordinaten umgerechnet haben, können Sie mittels DRAW den Punkt setzen.

4.3.1 Parallelprojektion

Jetzt, da wir nun bereits das Grundwissen über allgemeine Grafikprogrammierung besitzen, wollen wir uns mit einem schon viel komplexeren Problem beschäftigen, der dreidimensionalen Grafik. Sie selber haben bestimmt schon einmal, vorwiegend in Werbungen, eine 3-D-Grafik betrachten können, und müßten dann auch am besten wissen, was für eine Ausstrahlung hinter einer solchen Grafik steckt. Die Krönung wird natürlich durch die "4-D-Grafik", also die bewegte dreidimensionale Grafik erreicht. Größere Computer sind heute schon in der Lage, ganze Zeichentrickfilme in diesem Verfahren herzustellen.

Doch so hinaus wollen und können wir auch nicht. Noch nicht. Denn uns fehlen bis jetzt die theoretischen Vorraussetzungen um diese sehr ästhetische Art der Grafik zu programmieren.
Um nun auch unserem Computer diese Form der Grafik beizubringen bedarf es (leider) einiger Theorie, die ich Ihnen im folgenden beschreiben will:
Um ein Gebilde räumlich darzustellen, ist es mit dem herkömlichen Koordinatensystem nicht mehr getan, da wir in diesem nicht unterscheiden können, ob ein Punkt vorne, oder aber in weite Ferne gerückt ist. Wir müssen also eine Raumachse einführen, die üblich auch 'Z-Achse' heißt. Auf dieser Achse können wir nun auch die Raumkoordinaten eines Punktes festlegen:



Wie zeichnet man nun einen Punkt in ein solches Koordinatensystem ein? - Nun, zunächst einmal verfahren wir genau so, wie wir es von dem zweidimensionalen Koordinatensystem gewohnt sind. Danach zeichnen wir eine Parallele zur Z-Achse die durch den entstanden Punkt, nennen wir in 'p', verlaufen muß. Auf dieser Parallelen tragen wir nun den Abstand der Z-Koordinate ab, und erhalten unseren neuen dreidimensionalen Punkt (siehe Abb. 3.7.3).

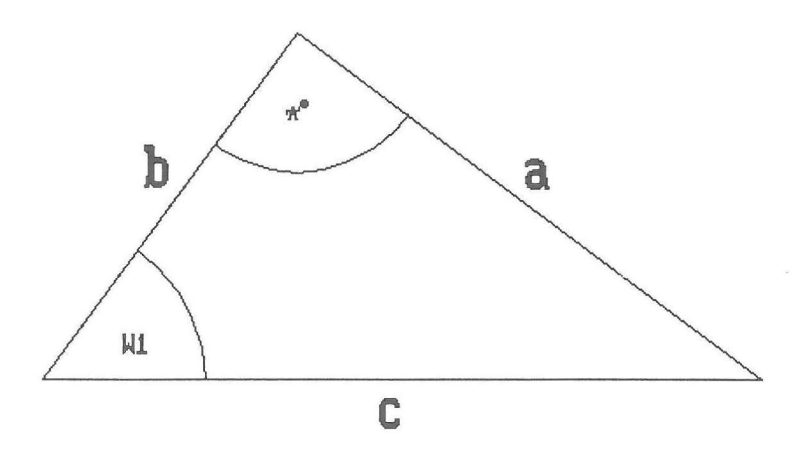
Das Problem, das sich ergibt, ist folgendes. Wir müssen einen Rechnenalgorithmus finden, der es uns ermöglicht einen Punkt, der in einem 3-D Koordinatensystem liegt, in das zweidimensionale Koordinatensystem zu übertragen.

Doch um dieses Problem erfolgreich angehen zu können, müssen wir uns zunächst etwas mit der Trigonometrie befassen:

Die Definition des Sinus

Die Definition des Sinus besagt, daß in einem rechtwinkligen Dreieck der Sinus des Winkels Alpha gleich dem Quotient von

Gegenkathete und Hypotenuse ist. Anhand der folgenden Zeichnung, können Sie sich diese Begriffe vergegenwärtigen:



Ausgegangen von w1, ist

a=Gegenkathete (denn a liegt gegenüber von w1)

b=Ankathete (denn b liegt an w1)

c=Hypotenuse

In diesem Fall wäre also $\sin(w1) = \frac{a}{c}$

Die Definition des Kosinus

Diese zweite Winkelfunktion erklärt nun ein weiteres Verhältnis in einem rechtwinkligen Dreieck. Sie besagt nämlich, daß der Kosinus von Alpha gleich dem Quotient von Ankathete und Hypotenuse ist. Um das zu verdeutlichen, dient ebenfalls die obrige Zeichnung.

Bezogen auf den Kosinussatz lautet unsere Gleichung:

```
          b
cos(wa)= ---
          c
```

Damit sie nun auch die Möglichkeit haben, verschiedene Parametereinstellungen auszuprobieren, haben wir einen 3-D-Designer, den Sie ja vielleicht schon aus dem Grafikbuch zum Commodore 64 kennen, auf den Commodore 128 übertragen:

```
100 rem    ********************
110 rem    **                **
120 rem    **  3-d-designer  **
130 rem    **                **
140 rem    ********************
150 rem
160 rem
270 rem
280 rem         *********
290 rem   ****  parameter  ***
300 rem         *********
310 rem
320 f1 = 5 : f2 = 5 : f3 = 3 : rem zerrung
330 ar = 15 : br =  0 : cr = 10 : rem verschiebung der raumkoord.
340 w =  /4: rem sichtwinkel (in rad)
350 si = sin(w) : co = cos(w) : rem konstanten
360 v1 = 100 : v2 = 180 : rem verschiebung der ebenenkoord.
370 rem
380 rem         **********
390 rem   ****  umrechnung  ***
400 rem         **********
410 rem
420 color 0,1    : rem hintergrundfarbe
430 color 1,6    : rem zeichenfarbe
440 color 4,1    : rem rahmenfarbe
```

```
450 graphic 1,1 : rem graphik init
460 fg=0:if v1<0 or v1>319 then fg=1:goto 480 : rem flag
470 draw ,v1,0 to v1,199 : rem raum-z-achse
480 if v2<0 or v2>200 then fg=1:goto 500
490 draw ,0,v2 to 319,v2 : rem raum-x-achse
500 z2 = v1 - (199-v2)/si*co : z1 = v1 + v2/si*co
510 if fg=0 and z1>=0 and z1<320 then draw ,v1,v2 to z1,0: rem raum-y-
achse oben
520 if fg=0 and z2>=0 and z2<320 then draw ,v1,v2 to z2,199: rem raum-y-
achse unten
530 read ap : dim x%(ap),y%(ap) : rem anzahl punkte
540 for za=1 to ap : rem punktezahl
550 read xr,zr,yr
560 x%(za) =  f1*(xr+ar) + f3*(yr+cr)*co+v1
570 y%(za) = -f2*(zr+br) - f3*(yr+cr)*si+v2
580 next za : rem naechster punkt
590 rem
600 rem          *******
610 rem    ****  striche  ***
620 rem          *******
630 rem
640 read al : rem anzahl linien
650 for za=1 to al
660 read p1,p2 : rem punktnummern lesen
670 if x%(p1)<0 or y%(p1)<0 or x%(p2)<0 or y%(p2)<0 then goto 700:rem
ausserhalb
680  ifx%(p1)>319ory%(p1)>199orx%(p2)>319ory%(p2)>199then  goto  700:rem
ausserhalb
690 draw ,x%(p1),y%(p1) to x%(p2),y%(p2) : rem verbinden
700 next za : rem naechste linie
710 getkey a$
720 graphic 0,1 : list : rem graphik aus
730 rem
740 rem      ***********
750 rem ****  koordinaten  ****
760 rem      ***********
770 data 10 : rem anzahl punkte
780 data  0, 0, 0,  6, 0, 0,  6,10, 0
790 data  0,10, 0,  3,15, 0,  3,15,15
800 data  6,10,15,  6, 0,15,  0, 0,15
```

```
810 data  0,10,15
820 rem
830 rem          *************
840 rem ****  verbindungen  ****
850 rem          *************
860 data 17 : rem anzahl linien
870 data  1, 2,  2, 3,  3, 4,  4, 1
880 data  4, 5,  5, 3,  5, 6,  6, 7
890 data  7, 3,  7, 8,  8, 2,  8, 9
900 data  9, 1,  9,10, 10, 4, 10, 6
910 data 10, 7
```

4.3.2 Zentralprojektion

Wenn wir Gegenstände genau betrachten, so können wir Ihre Entfernung unter anderem dadurch feststellen, daß sie, je weiter sie stehen, um so kleiner werden. Das klassische Beispiel dafür sind die Eisenbahnschienen, die immer schmaler werden und sich in weiter Ferne optisch treffen, was in der Realität natürlich nicht der Fall ist, denn die Schienen sind und bleiben parallel zueinander. Es ist tatsächlich so, daß alle Linien sich auf einen Punkt zuzubewegen scheinen. Dieser Punkt wird im allgemeinen Fluchtpunkt genannt. Das Problem für uns ist es nun, ein mathematisches Verfahren zu entwickeln, um die Linien auf einen Punkt zu richten. Die Herleitung der Formeln würde zu lange dauern. Die Scalierung soll mit (f1/f2/f3) und die Verschiebung mit (a/b/c) angegeben werden. Im folgenden wird ferner von einem orthogonalen dreidimensionalen Koordinatensystem ausgegangen. Dem Mathematiker wird dieser Begriff bekannt sein, denn es handelt sich um ein rechtwinkliges Koordinatensystem im dreidimensionalen Raum. Diese Rechtwinkligkeit bezieht sich auf die drei Achsen. Dabei soll die z-Achse die Raumachse sein. Wie bei der Parallel-Projektion müssen die drei Koordinaten eines Punktes in zwei umgewandelt werden:

```
x=(f1*xr+a)/q
y=(f2*yr+b)/q

mit  q=1-(f3*zr+c)/fpz
```

für:

xr,yr,zr: räumliche Koordinaten
f1,f2,f3: Verzerrung in xr-,yr-,zr-Richtung
a, b, c: Verschiebung in xr-,yr-,zr-Richtung
fpz : Entfernung (z-Koord.) des Fluchtpunktes

Sie sehen, daß Sie erst q errechnen müssen, um die beiden gesuchten Koordinaten zu ermitteln. Das Objekt, welches Sie zeichnen können Sie auch noch drehen. Es wird jedoch noch komplizierter:

```
x=(f1*(A*xr+D*yr+G*zr)+a)/q
Y=(f2*(B*xr+E*yr+H*zr)+b)/q

mit  q=1-(f3*(C*xr+F*yr+I*zr)/fpz
```

Diese drei Gleichungen sehen schon recht kompliziert aus, doch es konnt noch besser. Sie müssen für A,B,.....,C ja auch noch einsetzen. Dieses Einsetzen erfolgt nach folgenden Formeln:

```
A= cos(s)*cos(t)
B= sin(s)*cos(t)
C=-sin(t)
D=-sin(s)*cos(u)+cos(s)*sin(t)*sin(u)
E= cos(s)*cos(u)+sin(s)*sin(t)*sin(u)
F= cos(t)*sin(u)
G= sin(s)*sin(u)+cos(s)*sin(t)*cos(u)
H=-cos(s)*sin(u)+sin(s)*sin(t)*cos(u)
I= cos(t)*cos(u)
```

In diesen Formeln stecken aber wieder drei unbekannte Größen: s, t, u. Diese müssen wir also noch definieren:

s: Winkel der Rotation um die z-Achse
t: Winkel der Rotation um die y-Achse
u: Winkel der Rotation um die x-Achse

Diese Berechnungen sind schon sehr aufwendig. Daher ist es aus Zeitgründen ratsam, diese Berechnungen in Maschinensprache durchzuführen. Falls immer nur bestimmte Winkelwerte auftreten, wäre es außerdem sinnvoll, Tabellen anzulegen. Für diejenigen unter Ihnen, die der Maschinensprache nicht mächtig sind, möchten wir ein BASIC-Programm vorstellen, daß das Prinzip des 3-D-Designers übernimmt und nur auf die Zentralprojektion überträgt.

```
100 rem    ********************
110 rem    **                **
120 rem    ** 3-d-designer   **
130 rem    **                **
140 rem    ********************
150 rem
160 rem
170 rem
180 rem          *********
190 rem   ****   parameter  ***
200 rem          *********
210 rem
220 f1 = 10: f2 = 10: f3 = 6 : rem zerrung
230 ar = 15 : br =  0 : cr = 10 : rem verschiebung der raumkoord.
240 w =  /4: rem sichtwinkel (in rad)
250 si = sin(w) : co = cos(w) : rem konstanten
260 v1 = 100 : v2 = 180 : rem verschiebung der ebenenkoord.
270 fu = .0174532925
```

```
280 s = 0 : s = s * fu
290 t = 25: t = t * fu
300 u = 0 : u = u * fu
310 fp =-250
320 :
330 rem
340 rem           **********
350 rem    ****   umrechnung  ***
360 rem           **********
370 rem
380 color 0,1    : rem hintergrundfarbe
390 color 1,6    : rem zeichenfarbe
400 color 4,1    : rem rahmenfarbe
410 graphic 1,1 : rem graphik init
490 read ap : dim x%(ap),y%(ap) : rem anzahl punkte
500 for za=1 to ap : rem punktezahl
510 read xr,yr,zr
520 :
530 a=cos(s)*cos(t)
540 b=sin(s)*cos(t)
550 c=-sin(t)
560 d=-sin(s)*cos(u)+cos(s)*sin(t)*sin(u)
570 e=cos(s)*cos(u)+sin(s)*sin(t)*sin(u)
580 f=cos(t)*sin(u)
590 g=sin(s)*sin(u)+cos(s)*sin(t)*cos(u)
600 h=-cos(s)*sin(u)+sin(s)*sin(t)*cos(u)
610 i=cos(t)*cos(u)
620 :
630 q=1-(f3*(c*xr + f*yr + i*zr)+cr)/fp
640 :
650 x%(za) = ((f1*(a*xr + d*yr + g*zr)+ar)/q)+v1
660 y%(za) =-((f2*(b*xr + e*yr + h*zr)+br)/q)+v2
670 :
710 next za : rem naechster punkt
720 rem
730 rem           *******
740 rem    ****   striche  ***
750 rem           *******
760 rem
770 read al : rem anzahl linien
```

```
780 for za=1 to al
790 read p1,p2 : rem punktnummern lesen
800 if x%(p1)<0 or y%(p1)<0 or x%(p2)<0 or y%(p2)<0 then goto 830:rem
ausserhalb
810  ifx%(p1)>319ory%(p1)>199orx%(p2)>319ory%(p2)>199then  goto  830:rem
ausserhalb
820 draw ,x%(p1),y%(p1) to x%(p2),y%(p2) : rem verbinden
830 next za : rem naechste linie
840 getkey a$
850 graphic 0,1 : list : rem graphik aus
860 rem
870 rem          ***********
880 rem ****  koordinaten  ****
890 rem          ***********
900 data 10 : rem anzahl punkte
910 data  0, 0, 0,  6, 0, 0,  6,10, 0
920 data  0,10, 0,  3,15, 0,  3,15,15
930 data  6,10,15,  6, 0,15,  0, 0,15
940 data  0,10,15
950 rem
960 rem          ************
970 rem ****  verbindungen  ****
980 rem          ************
990 data 17 : rem anzahl linien
1000 data  1, 2,  2, 3,  3, 4,  4, 1
1010 data  4, 5,  5, 3,  5, 6,  6, 7
1020 data  7, 3,  7, 8,  8, 2,  8, 9
1030 data  9, 1,  9,10, 10, 4, 10, 6
1040 data 10, 7
```

V. Anwendungen der Grafikprogrammierung

5.1 Unterhaltungsgrafik

5.1.1 Laufschriften

Die zunehmende Beliebtheit der Computergrafik machte man sich nicht nur in der Mathematik, Statistik oder Konstruktionszeichnung zunutze. Nein, auch ein Zweig, von dem man glaubte, er habe nichts mit Computern zu tun, bedient sich dieser Art der Grafikdarstellung: die Werbung. Man nehme nur die Leucht-Band-Reklamen, die schon in fast jedem Schaufenster zu sehen sind. Natürlich sollten Sie auch Ihren Computer für solche Dinge nutzen können. Warum also nicht auch eine schöne Laufschrift auf den Bildschirm des Commodore 128 zaubern. Diesem Problem wollen wir uns nun im folgenden widmen. Das Prinzip beruht darauf, Zeichen in vergrößerter Form auf den Bildschirm zu bringen, und den ganzen Bildschirm dann nach rechts zu scrollen, um weiteren Buchstaben oder Ziffern die Möglichkeit zu geben, auf dem Bildschirm erscheinen zu können.
Das folgende Programm, das mehr durch seine Inkomplexität als durch seine Schnelligkeit verblüfft, führt nun eben diese Tätigkeit aus, wobei ein Teil der schon bekannten Scrollroutine aus dem VIC-Kapitel übernommen wurde.

```
100 rem ******************************
110 rem ***                        ***
120 rem ***     laufschriften      ***
130 rem ***                        ***
140 rem ******************************
150 :
160 color 0,1:color 4,1
170 gosub 410              : rem assemlber-programm einlesen
180 dim ch(70)             : rem maximale anzahl an zeichen
190 bank 14                : rem chargenerator zugaenglich machen
```

```
200 graphic 0,1           : rem 40 zeichengrafik einschalten
210 input "bitte text eingeben";t$
220 char ,0,0,t$          : rem text oben, links auf den bildschirm
230 :
240 for z=1 to len(t$)    : rem bildschrim-code uebernehmen
250 :ch(z)=peek(1023+z)
260 next z
270 :
280 scnclr
290 :
300 for z=1 to len(t$)
310 :ad=53248+8*ch(z)
320 ::for x=7 to 0 step-1  : rem 8 bits
330 ::for xx=0 to 7        : rem 8 bytes
340 :::if (peek(ad+xx)and2^x)=2^x then char ,38,10+xx,chr$(209)
350 ::next xx:sys 3072     : rem scrollen
360 next x,z
370 :
380 for x=0 to 38:sys 3072:next
390 goto 300
400 :
410 rem einlesen der datas
420 :
430 for x=0 to 61
440 :read a$:a=dec(a$)
450 :poke dec("c00")+x,a
460 :p=p+a
470 next x
480 :
490 if p<>9543then print "fehler in datas":end
500 return
510 :
520 rem assembler-datas
530 :
540 data a9,00,a0,04,85,fa,84,fb
550 data a9,00,a0,d8,85,fc,84,fd
560 data a0,00,a2,14,c8,b1,fa,48
570 data b1,fc,88,91,fc,68,91,fa
580 data c8,c0,27,d0,ef,a9,20,91
590 data fa,a5,fa,18,69,28,85,fa
```

```
600 data 85,fc,90,04,e6,fb,e6,fd
610 data a0,00,ca,10,d7,60
```

5.1.2 Sprite-Animation

Aber nicht nur solche Sachen sind von großer Bedeutung für die Werbung. Stellen Sie sich doch einmal einen Computer vor, der sich selber anpreist. Dieses Darbieten muß natürlich in einem sehr sauberen Stil erfolgen. Ein Männchen, das z. B. über den Bildschrim wandert, sollte möglichst in vielen Einzelbewegungen und natürlich auch flimmerfrei bewegt werden!
Wenn von einer "Animation" gesprochen wird, so ist das natürlich nichts anderes. Jeder Zeichentrickfilm ist z. B. eine solche Animation. Diese kann sehr gut mit Sprites realisiert werden, dessen Aussehen sich sinnvoll hintereinander ändert. In dem folgenden Programm wurde eine solche Animation realisiert:

```
100 rem ****************************
110 rem ***                      ***
120 rem ***  sprite-animation    ***
130 rem ***  ----------------    ***
140 rem ***                      ***
150 rem ****************************
160 :
170 for x=0 to 255        : rem 4*64=256 bytes lesen
180 :read a$
190 :poke dec("e00")+x,dec(a$)
200 next x
210 :
220 sprsav 1,a$(1)       : rem die vier verschiedenen
230 sprsav 2,a$(2)       : rem spritedefinitionen,
240 sprsav 3,a$(3)       : rem in einem ein-dimensionalen
250 sprsav 4,a$(4)       : rem array speichern
260 :
```

```
270 sprite 1,1,6,1,1,1,1 : rem sprite einschalten
280 sprcolor 2,3         : rem multi-farben definieren
290 movspr 1,160,100     : rem sprite positionieren
300 :
310 movspr 1,90#8        : rem sprite bewegen
320 :
330 for x=1 to 4
340 sprsav a$(x),1
350 for p=1 to 10:next
360 next
370 goto 330
380 :
390 rem *** sprite datas ***
400 :
410 data 00,00,00,00,00,00,00,00,00,00,00,00,01,55,55,00
420 data 01,00,00,01,00,00,05,50,40,15,f4,50,15,f5,55,55
430 data 55,55,55,55,00,05,94,00,01,50,00,01,11,01,55,54
440 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
450 data 00,00,00,00,00,00,00,00,00,40,00,00,15,50,00
460 data 01,04,00,01,00,04,05,50,50,15,f4,50,15,f5,55,55
470 data 55,55,55,55,04,05,94,00,01,50,00,01,11,01,55,54
480 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
490 data 00,00,00,00,00,00,00,00,00,10,00,00,05,40,00
500 data 01,10,00,01,00,10,05,50,50,15,f4,50,15,f5,55,55
510 data 55,55,55,55,10,05,54,00,01,50,00,01,11,01,55,54
520 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
530 data 00,00,00,00,00,00,00,00,00,40,00,00,15,50,00
540 data 01,04,00,01,00,04,05,50,50,15,f4,50,15,f5,55,55
550 data 55,55,55,55,04,05,54,00,01,50,00,01,11,01,55,54
560 data 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

Im Prinzip kann das gleiche auch mit hochaufgelösten Shapes erfolgen, jedoch ist der Aufbau dieser durch sehr komplizierte Berechnung derselben ungleich schwieriger.

5.1.3 Etwas über Spiele

Nun noch ein letztes Thema, das mit der Unterhaltungsgrafik zu tun hat, und das wir Ihnen ebenfalls noch vorstellen möchten. Es sind die Spiele.
Während die Microelektronik immer weiter fortschritt, wuchs auch gleichzeitig ein neuer Branchenzweig: Die Videospiel-Branche. Zu Beginn dieser Entwicklung war ein Videospiel, damals nannte man es "Telespiel", noch sehr primitiv gestaltet: Ein Balken war die Spielfigur und damit mußte man mit einem Ball spielen. Doch mit der Zeit wurden die Videochips immer wieder besser und damit wuchs auch die maximale Auflösung des Spielfeldes. Diese erweiterte Auflösung kam den Programmierern sehr gelegen, konnten sie doch nun ihre ganze Phantasie grafisch gut umsetzen. Aus den Balken wurden Männchen, aus dem starren Spielfeld wurde ein flexibles. Heute fährt man mit einem Auto über eine ziemlich realistische Fahrbahn, oder man jagt mit einem Raumschiff durch Höhlen.

Bei den Spielen mit dem Computer gibt es drei auf die Grafik bezogene Kategorien:

1. Ohne Grafik
2. Feste Grafik
3. Sich bewegende Grafik

Die erste Kategorie bezieht sich auf Strategie- oder Denkspiele. Diese Spiele gehören heute aber auch schon zur zweiten Kategorie, neben einigen Formen von Adventures. In den Spielen, bei denen die Handlung mit dem Joystick oder den Paddels zählt, ist eine sich bewegende Grafik unerläßlich. Ohne die Bewegung auf dem Bildschirm wäre ein Spiel schnell langweilig. Nun, die Softwarefirmen, die solche Spiele vertreiben, haben zumeist gute Programmierer unter Vertrag, so daß man sich wirklich gute Spiele kaufen kann. Doch nach einigen Spielen verspürt man selbst den Drang zum do-it-yourself, zum Selberprogrammieren. In diesem Grafikbuch sollten Sie schon einiges Werkzeug hierzu erhalten haben. So müßten Sie in der Lage sein, Sprites selber zu handhaben. Sprites sind sehr wichtige

Elemente einer bewegten Grafik. Dennoch sollten Sie vorsichtig damit umgehen, denn Sie haben ja nur 8 verschiedene Sprites zur gleichen Zeit anwendungsbereit.

Betrachtet man das schon erwähnte Autospiel genauer, so bewegt sich doch das Auto meistens nicht, jedenfalls nicht wesentlich. Die Bewegungen beschränken sich doch im Großteil aller Fälle nur auf rechts oder links. Der wichtigste Bewegungsvorgang erfolgt durch das Scrollen des Bildschirms. Diesen Vorgang kann man mit den in den Kapiteln über das Scrolling stehenden Programmen erreichen.

Doch warum gleich in die tiefe Trickkiste greifen, wenn es einfacher auch geht? Ein fester Hintergrund kann absolut genügen. Wenn dazu noch ein oder mehrere Sprites die Bewegungen ausführen, ist es schon möglich, selbst hochinteressante Spiele herzustellen. Wichtig dabei ist die originelle Spielidee, denn wer möchte schon mit der hundertsten Version von Pac-Man oder Space-Invaders spielen. Der Spielwitz ist sehr entscheidend. Auch originell, auffallend sollte das selbsterzeugte Spiel sein. Man erinnere sich: Es gibt Spiele, die zum Großteil wegen ihrer musikalischen Untermalung hochgelobt wurden. Die Spielidee oder ihre Ausführung ist dagegen eintönig. Also: Auch die akustische Untermalung eines Spiels sollte vorhanden sein. Die genauere Durchführung dieses Themas können Sie in einem anderen Fachbuch erfahren.

Das Auffallen eines Spiels sollte allerdings nicht auf geschmacklose Art und Weise erreicht werden, da diese Geschmacklosigkeit nach einiger Zeit nicht mehr anziehend, sondern abstoßend wirkt. Neben den Spielen, die ihren Inhalt der Phantasie entnehmen, sind auch Spiele aus dem Alltag sehr interessant. So könnte ich mir vorstellen, daß die Parkplatzsuche in einer Großstadt eine gute Idee für ein Spiel ist. Der Hintergrund besteht aus Häuserblöcken und Straßen. Am einfachsten wäre eine Stadt im amerikanischen Stil: Die Sraßen sind wie ein Raster aufgebaut. Dazwischen sind die Häuser. Auf den Straßen müssen mehr oder weniger Parklücken sein. Interessant wäre es auch, noch Einbahnstraßen in das Straßennetz einzubauen. Soweit für den Hintergrund. Die Spielfigur, ein Auto, kann man

mit Hilfe eines Sprites definieren. Die parkenden Autos werden mit der Shape-Grafik definiert und in das Straßennetz eingebaut. Wenn ein parkendes Auto einen Parkplatz räumt, so muß es vom Spielfeld verschwinden. Dazu benutzen wir wieder die Sprite-Grafik. Neben unserem Auto sind noch Gegenspieler nötig, also Autos, die auch einen Parkplatz suchen. Diese Autos werden auch als Sprites definiert. Am besten ist es, wenn die Gegenspieler und die schon parkenden Autos die gleiche Form haben und nur unseres hat eine andere. Schon haben wir einen einfachen grafischen Aufbau für ein Spiel, das einen richtigen Realitätswert hat und alle Autoanfänger seelisch auf die kommenden Jahre mit ihrem Auto vorbereitet; so kam mir diese Idee, als ich in meinem Wohnort wieder einmal lange nach einem Parkplatz suchte und, welch ein Wunder, ich hatte auch einen gefunden. Die logischen Operationen sind bei diesem Spiel nicht so schwer. Die interne Uhr wird für die Länge der Parkplatzsuche verwendet. Je kürzer die Suche, desto besser. Die Kontrolle über das Auto sollte über einen Joystickport erfolgen und die Auswahl der Autos, die einen Platz frei machen, erfolgt zufällig.

Versuchen Sie doch selbst einmal, solch ein Spiel zu verwirklichen, da die Grundlagen doch nun vorhanden sein sollten. Lassen Sie Ihrer Phantasie freien Lauf, und sehen Sie, welch gute Spielidee sich hinter noch so einer trivialen und alltäglichen Situation verstecken kann!

5.2 Statistik

Grundsätzliches zur Statistik

Wir kommen nun zu einer der interessantesten und nützlichsten Anwendungen der Computergrafik. Gemeint ist die Statistik. Unter Statistik versteht man die zahlenmäßige Zusammenfassung von Ergebnissen (Werten), meist in Form von Tabellen, aber auch in Form anschaulicher Grafiken. Wir werden uns in den

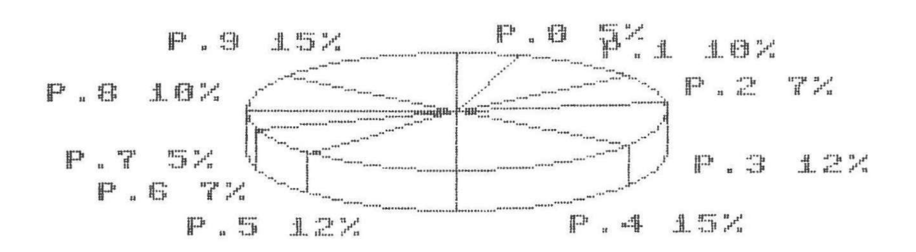
folgenden Kapiteln ausschließlich auf die beschreibende (deskriptive) Statistik beschränken, da diese speziell die Beschreibung von gegebenem Zahlenmaterial umfaßt. In der grafischen Statistik wird grundsätzlich zwischen vier verschiedenen Darstellungsarten unterschieden:

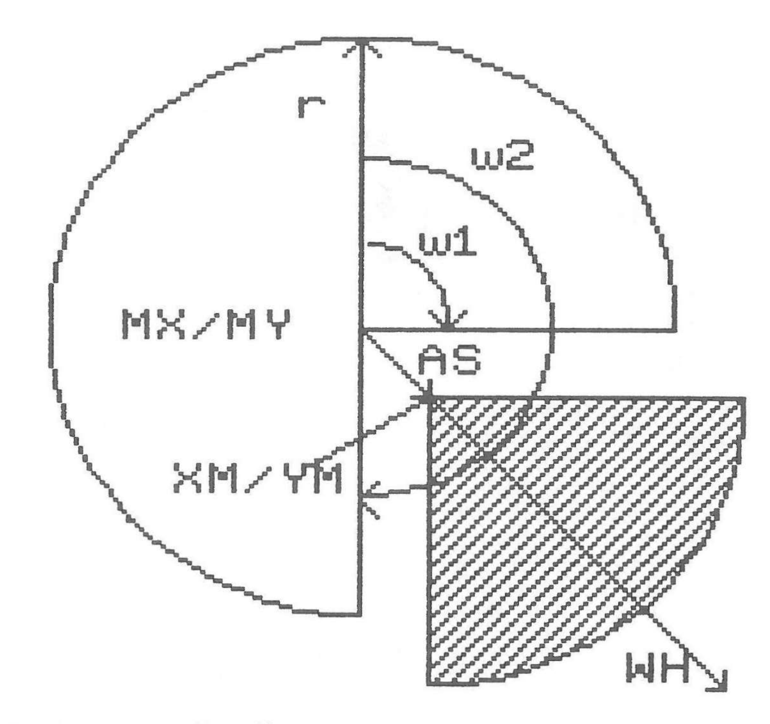
a) Kreis- und Kuchendiagramme
b) Säulen und Balkendiagramme
c) Kurvenstatistiken
d) Vergleichsgrafiken

a) Kreis bzw. Kuchendiagramme:

Kreis und Kuchendiagramme beschreiben, entgegen den anderen drei Darstellungsformen, nicht Zahlenmaterialien über einen bestimmten Zeitraum, sondern Anteile einer Gesammtmenge (100%). Beispiele wären die Sitzeverteilung im Bundestag oder die Konzentration verschiedener Schwermetalle in Autoabgasen. Besonders die Übersichtlichkeit von Kuchendiagrammen ist ein großer Vorteil dieser Darstellungsform, da man, ganz im Gegensatz zu Tabellen mit prozentualen Angaben, hier immer alle Informationen auf einen Blick erfassen kann. Der einzige schwerwiegende Nachteil dieser "Kuchen" ist nur, daß sie relativ kompliziert zu berechnen sind, was aber anhand der Formeln in den nächsten Kapiteln für Sie ein Kinderspiel sein dürfte.

DEMONSTRATION KUCHENGRAFIK





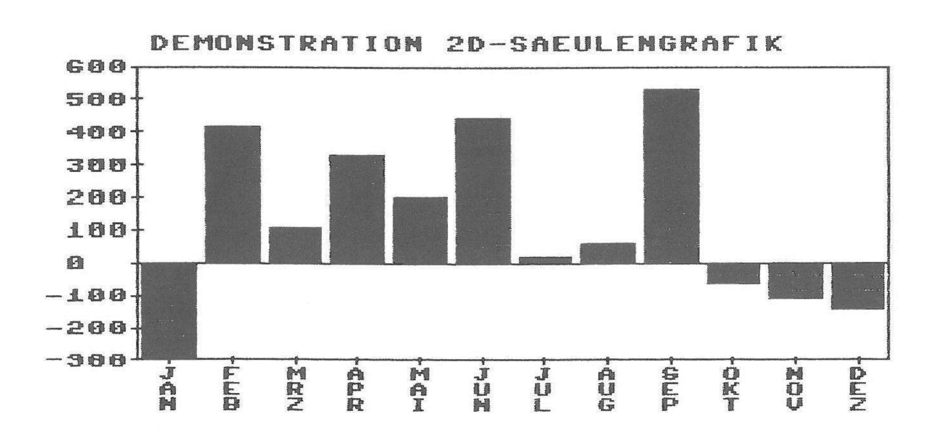
b) Säulen und Balkendiagramme:

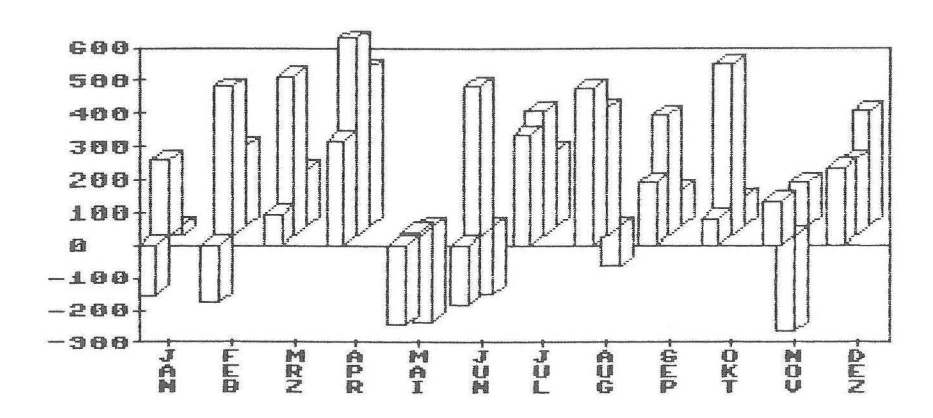
Säulen- und Balkendiagramme gehören wohl mit den Kuchendiagrammen zu den bekanntesten Möglichkeiten der grafischen Statistik. Kein Wunder, denn sie sind äußerst benutzerfreundlich und daher auch leicht zu lesen. Durch die Überschrift, die Achsenbeschriftungen und die Achseneinteilungen kann auch der Laie mit diesen Grafiken etwas anfangen.

Wir unterscheiden in diesem Buch zwei Arten der Säulendiagramme:

- Die 2-D-Diagramme
- Die 3-D-Diagramme

Hierbei stellen die 3-D-Diagramme eigentlich nur eine Erweiterung der 2-D-Diagramme dar, da sie sich praktisch nur durch die Anzahl der hintereinander liegenden Säulen unterscheiden.

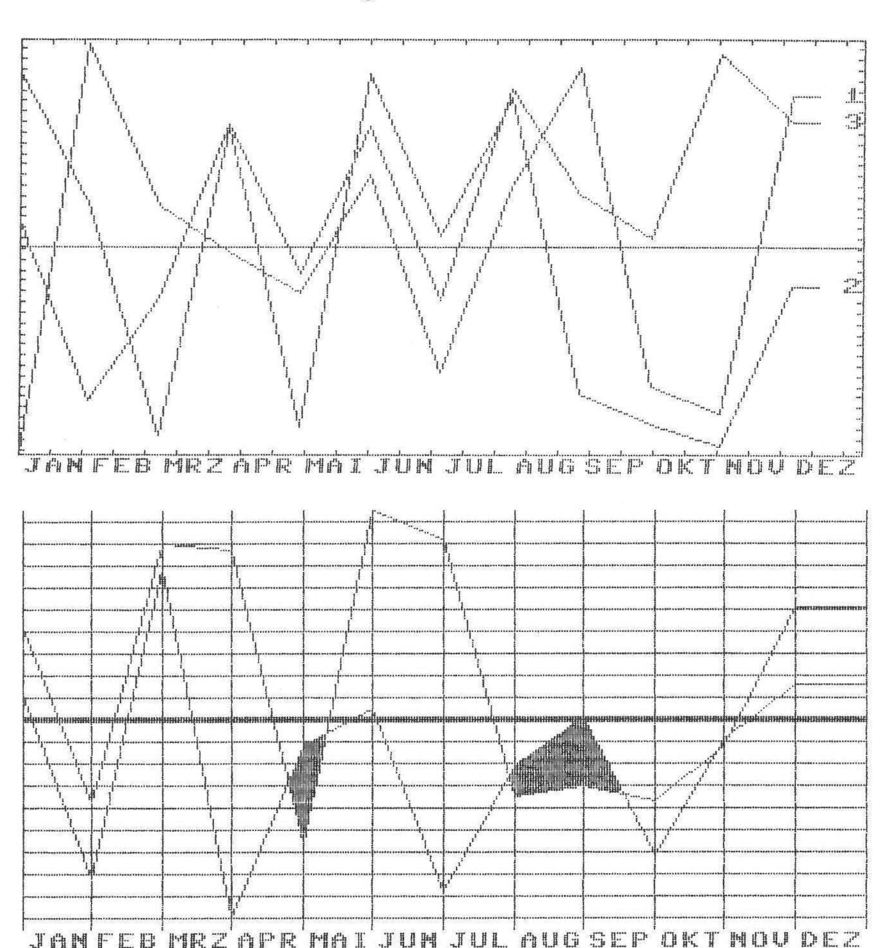




c) Kurvenstatistiken:

Kurvenstatistiken kennen wir wahrscheinlich alle noch aus dem Mathematik- oder Physikunterricht. Eigneten sie sich doch z. B. hervorragend zum Darstellen einer Weg/Zeit-Funktion in der Physik oder zur grafischen Annäherung einer Sinuskurve in der Mathematik. Die Anzahl der Werte, die verarbeitet werden können, liegt um einiges höher als bei den anderen Möglichkeiten der grafischen Darstellung. Dies hat zur Folge, daß sie z. B. bei

besonders großen Informationsmengen oder aber auch bei Meßreihen oder ähnlichem eingesetzt werden.



d) Vergleichsgrafiken:

Eigentlich stellt ja schon die dreidimensionale Säulengrafik eine Vergleichsgrafik dar, aber wir wollen in diesem Buch unter dem Begriff 'Vergleichsgrafiken' eine spezielle Form der Kurvengrafik behandeln. Der Unterschied zwischen beiden liegt darin, daß die normale Kurvengrafik (fast) beliebig viele Kurven umfassen kann, während die Vergleichsgrafik nur durch zwei Kurven gebildet wird, deren Differenzbereiche je nach Kurvenverlauf gefärbt werden.

5.2.1 Die 2-D-Balkengrafik

Nachdem wir nun schon in den vorangegangenen Kapiteln die Grundlagen der Grafikprogrammierung und die Hardwaregrundlagen besprochen haben, wollen wir jetzt auf die Anwendungsmöglichkeiten der Computergrafik näher eingehen. In diesem Buch behandeln wir die Statistik und die Darstellung von mathematischen Formeln mittels der Computergrafik. Beginnen wollen wir das Oberkapitel 'Statistik' mit der Balkengrafik. Balkengrafiken kennen Sie wahrscheinlich schon aus Wahlstatistiken oder ähnlichem, da sie sehr häufig angewendet werden, weil sie einfach zu bedienen und zu programmieren sind. Doch entpuppt sich die Programmierung als gar nicht so einfach, da wir gegebenenfalls auch noch alle Werte so anpassen müssen, daß sie überhaupt in unsere Grafik passen.

Noch etwas Allgemeines: Alle Statistikprogramm in diesem Buch sind absichtlich so allgemein gehalten, daß sie ohne Probleme auch auf andere Rechner angepaßt werden können. Oft reicht es schon, die Bildschirmgrenzen zu verändern, die immer am Anfang eines Programms in den Variablen XL, XH,YL und YH definiert sind. Probieren Sie doch einmal aus, wie die einzelnen Statistiken verkleinert aussehen oder schreiben Sie die Programme doch auf den 80-Zeichenschirm um. Eine komfortable Befehlserweiterung zur Grafikprogrammierung im 80-Zeichenmodus finden Sie im Kapitel über den VDC (Das ist der 80-Zeichenvideocontroller!). Doch nun etwas spezieller zu unserem Programm und seinen Anwendungsbereichen. Das Programm eignet sich zum Beispiel, um die Einnahmen einer Firma über ein Jahr grafisch auszuwerten. Sollte einer der eingegeben Werte

nicht mehr im Bereich von -300 bis +600 liegen, so macht dies nichts, denn das Programm gleicht dann alle Werte so an, daß auch die Extremwerte in die Grafik passen. Nach dem Programmstart verlangt das Programm von Ihnen 12 Werte und die Überschrift, die Sie nach eigenen Bedürfnissen festlegen sollten. Allerdings sollte sie nicht zu lang sein, da bei der Überschrift nicht geprüft wird ob die Bildschirmbreite ausreicht. Ist die Überschrift also zu lang, so kommt es zu einer Fehlermeldung. An dieser Stelle nun das Programm und weiter unten die Kurzbeschreibung, aus der Sie alle wichtigen Fakten entnehmen können sollten:

Programmlisting:

```
100 rem ***************************
110 rem *   2d-balkendiagramm     *
120 rem *  (12-monatsdiagramm)    *
130 rem ***************************
140 :
150 xl=  0 : xh=319 : rem x-grenzwerte
160 yl=  0 : yh=199 : rem y-grenzwerte
170 :
180 v1=xl+35 :       rem 1. vertikale
190 v2=xh :          rem 2. vertikale
200 :
210 h1=yl+15 :          rem 1.horizontale
220 h2=yh-31 :          rem 2.horizontale
230 h3=h2-(h2-h1)/3 : rem 3.horizontale
240 :
250 sb=((v2-v1)/24)-2 : rem saeulenbreite
260 :
270 color 0,1 : color 4,1 : color 5,14 : rem farben setzen
280 color 1,15 : scnclr : rem bildschirm loeschen
290 :
300 dim we (12),mo$ (12) : rem speicher reservieren
310 :
320 data jan,feb,mrz,apr,mai,jun
330 data jul,aug,sep,okt,nov,dez
340 :
```

```
350 for i=1 to 12 : read mo$ (i) : next : rem monate lesen
360 :
370 input "ueberschrift";s$ : rem ueberschrift holen
380 :
390 for i=1 to 12
400 :   print "werte fuer ";mo$ (i);". ";
410 :   input we (i) : rem wert einlesen
420 :   if we (i)>max then max=we (i) : rem maximum ?
430 :   if we (i)<min then min=we (i) : rem minimum ?
440 next i
450 :
460 rem **********************
470 rem *   achsen zeichen   *
480 rem **********************
490 :
500 fast : graphic 1,1 : rem grafik an, fast-modus
510 :
520 ft=(h3-h1)/600 : rem normal-faktor
530 if max<=600 and min>=-300 then 580 : rem innerhalb ?
540 ft=(h3-h1-1)/abs(max) : rem neuer faktor
550 if h3+abs (min)*ft <=h2 then 580 : rem minimum auch ok ?
560 ft=(h2-h3)/abs(min) : rem neuer faktor
570 :
580 draw 1,v1,h1 to v2,h1 : rem 1. horizontale zeichnen
590 draw 1,v1,h2 to v2,h2 : rem 2. horizontale zeichnen
600 draw 1,v1,h3 to v2,h3 : rem 3. horizontale zeichnen
610 draw 1,v1,h1 to v1,h2 : rem 1. vertikale zeichnen
620 draw 1,v2,h1 to v2,h2 : rem 2. vertikale zeichnen
630 :
640 z=0 : rem wert auf null
650 for i=h3 to h1 step -(h3-h1)/6
660 :   x=xl : y=i-3 : t$=str$ (z)
670 :   gosub 1050 : rem wert ausgeben
680 :   draw 1,v1-2,i to v1+2,i : rem einteilungen setzen
690 :   z=z+100 : rem wert erhoehen
700 next i
710 :
720 z=0 : rem wert auf null
730 for i=h3 to h2+1 step (h2-h3)/3
740 :   x=xl : y=i-3 : t$=str$ (z)
```

```
750 :   gosub 1050 : rem wert ausgeben
760 :   draw 1,v1-2,i to v1+2,i : rem einteilungen setzen
770 :   z=z-100 : rem wert erniedrigen
780 next i
790 :
800 for i=1 to 3 : z=1
810 :   for j=v1+sb+3 to v2 step (v2-v1)/12
820 :       x=j-4 : y=h2+4+((i-1)*8)
830 :       t$=mid$ (mo$ (z),i,1)
840 :       gosub 1050 : rem wert ausgeben
850 :       draw 1,j,h2+2 to j,h2-2 : rem einteilungen
860 :       z=z+1 : rem index erhoehen
870 :   next j
880 next i
890 :
900 char 1,(xl+(xh-xl)/2-(len (s$)*4))/8+1,yl/8,s$
910 :
920 slow : rem in slow-modus gehen
930 :
940 rem ***********************
950 rem *   werte eintragen   *
960 rem ***********************
970 :
980 x=1 : rem index auf eins setzen
990 for i=v1+sb+3 to v2 step (v2-v1)/12
1000 :   box 1,i-sb,h3-(we(x)*ft),i+sb,h3,0,1
1010 :   x=x+1 : rem index erhoehen
1020 next i
1030 goto 1030 : rem programmende
1040 :
1050 rem ******************************
1060 rem *   text (x,y,t$) ausgeben   *
1070 rem ******************************
1080 :
1090 xa=(xl and 248)+8 : ya=(yl and 248)+8
1100 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
1110 char 1,xa/8,ya/8,t$
1120 sshape z$,xa,ya,xa+len (t$)*8-1,ya+7
```

```
1130 gshape q$,xa,ya
1140 gshape z$,x,y
1150 return
```

Programmbeschreibung:

100 - 130:	Programmkopf.
140 - 160:	Definition der Bildschirmrandkoordinaten.
170 - 230:	X- bzw. Y-Koordinaten der einzelnen Achsen.
240 / 250:	Berechnung der Säulenbreite aus den Bildschirmgrenzen.
260 - 280:	Farben setzen und den Textbildschirm löschen.
290 / 300:	Speicher für die darzustellenden Werte reservieren.
310 - 330:	Datas für Monatsbezeichnungen.
340 / 350:	Monatsabkürzungen in Strings einlesen.
360 - 410:	Überschrift und Werte einlesen.
420 / 430:	Testen, ob der eingegebene Wert größer als das Maximum oder kleiner als das Minimum ist. Ist dies der Fall, wird das Maximum bzw. das Minimum gleich dem eingegeben Wert gesetzt.
440:	Schleifenende.
450 - 480:	Unterprogrammkopf.
490 / 500:	Fast-Modus aktivieren und HI-RES-Grafik aktivieren und löschen.
510 / 520:	Defaultwert für Y-Zerrfaktor, wenn alle Werte innerhalb des Bereichs von -300 bis +600 liegen.
530:	Testen ob das Minimum und das Maximum im erlaubten Bereich liegen. Wenn ja, wird der in Zeile 520 berechnete Defaultwert beibehalten und zur Zeile 580 gesprungen.

540 - 560:	War ein Wert nicht erlaubt, so wird der neue Umrechnungsfaktor zuerst für den positiven Bereich berechnet (540). Dann (550) wird getestet ob auch das Minimum bei diesem Faktor innerhalb des erlaubten Bereichs liegt. Ist dies nicht der Fall, so wird der neue Umrechnungsfaktor anhand des negativen Bereichs und des Minimums berechnet (560). Dieser Umrechnungsfaktor stimmt dann in jedem Fall.
570 - 620:	Einzeichnen der horizontalen und vertikalen Achsen.
630 - 700:	Einteilungen an der vertikalen Achse eintragen und Werte mittels der Unterroutine ab Zeile 1050 ausgeben. Der Wertebereich liegt in dieser Schleife von 0 bis +600.
710 - 780:	Anteilungen an der vertikalen Achse eintragen, allerdings diesmal für den negativen Teil von 0 bis -300.
800 - 880:	Einteilungen an der horizontalen Achse antragen und Monatsabkürzungen untereinander mittels der Textausgaberoutine (ab 1050) ausgeben.
890 / 900:	Zentrierte Ausgabe der Überschrift.
910 / 920:	Zurückschalten in den SLOW-Modus, damit das Einzeichnen der Säulen sichtbar ist.
930 - 960:	Unterprogrammkopf.
970 - 1020:	Einzeichnen der Balken anhand der umgerechneten Werte (*FT).
1030:	Programmende.
1040 - 1070:	Unterprogrammkopf der Textausgaberoutine.
1080 - 1090:	Definieren der glatt durch acht dividierbaren X- und Y -Koordinaten.
1100:	Speichern des Bereiches, in dem der Text zwischengespeichert wird im String Q$.
1110:	Ausgeben des Textes an den in Zeile 1090 berechneten Koordinaten.
1120:	Speichern des Textes im String Z$.
1130:	Zurückschreiben des Strings Q$ in den Puffer.
1140:	Positionieren des eigentlichen Textes an den angegebenen Koordinaten X,Y.
1150:	Rücksprung in das Hauptprogramm.

5.2.2 3-D-Balkengrafik

Nachdem wir nun schon die erste, einfachere Form der Balkengrafik, nämlich die 2-D-Balkengrafik, besprochen haben, wollen wir Ihnen an dieser Stelle die dreidimensionale Balkengrafik vorstellen. Dreidimensional deshalb, weil jeweils maximal drei Säulen hintereinander liegen. Hierdurch wird ein räumlicher Effekt hervorgerufen. Aber diese Darstellungsart hat auch seine Nachteile:

Sie stellt unserer Meinung nach die schwierigste Form der grafischen Statistik dar, da sie recht aufwendig zu programmieren ist. So kommt es beim Zeichnen der Grafik manchmal zu Überschreitungen der obersten horizontalen Achse, was jedoch die ordnungsgemäße Funktionsweise des Programms in keinster Weise stören sollte. Aufgrund der Hintereinanderlagerung von maximal drei Säulen ergibt sich zwangsläufig, daß man die Grafik nicht ohne Probleme ablesen kann. Beachten Sie also bitte, daß die räumliche Anordnung der Säulen in der Skala nicht berücksichtigt werden konnte. Ansonsten gibt es zu diesem Programm nichts zu sagen, da fast alles, bis auf die Zeichnung der Säulen, mit der 2-D-Balkengrafik identisch ist. Nur die Anwendungsbereiche der beiden Statistikarten unterscheiden sich etwas voneinander. So ist es mittels der 3-D-Balkengrafik, im Gegensatz zur 2-D-Balkengrafik, möglich, Vergleiche zwischen mehreren Jahren zu ziehen und so Unterschiede festzustellen. Doch nun genug der Vorrede! Hier das Programm:

Programmlisting:

```
100 rem ***************************
110 rem *   balkendiagramm 3d     *
120 rem *  (12-monatsdiagramm)    *
130 rem ***************************
140 :
```

```
150 xl=  0 : xh=319 : rem x-grenzwerte
160 yl=  0 : yh=199 : rem y-grenzwerte
170 :
180 v1=xl+35 : rem 1. vertikale achse
190 v2=xh :    rem 2. vertikale achse
200 :
210 h1=yl+15 :          rem 3. horizontale achse
220 h2=yh-31 :          rem 2. horizontale achse
230 h3=h2-(h2-h1)/3 : rem 3. horizontale achse
240 :
250 sb=((v2-v1)/(3.5*12)) : ts=sb : rem saeulenbreite/tiefe
260 :
270 w=270+46 : rem winkel gleich 45 grad
280 :
290 def fnx(x)=xr+yr*cos (w*fa) : rem umrechnung x
300 def fny(y)=zr+yr*sin (w*fa) : rem umrechnung y
310 :
320 fa=3.1415/180 : rem umrechnungsfaktor grad -> radiant
330 :
340 color 0,1 : color 4,1 : color 5,14 : rem farben setzen
350 color 1,15 : scnclr : rem textbildschirm loeschen
360 :
370 input "wieviel reihen";ar : rem anzahl reihen einlesen
380 if ar<1 or ar>3 then 370 : rem erlaubt ?
390 :
400 dim we(ar,13),mo$(13) : rem speicher reservieren
410 :
420 data jan,feb,mrz,apr,mai,jun
430 data jul,aug,sep,okt,nov,dez
440 :
450 fori=1to12 : read mo$(i) : next : rem monate einlesen
460 :
470 for i=1 to ar : rem reihen
480 :   for j=1 to 12
490 :       print"werte fuer reihe";i;mo$(j);". ";
500 :       input we(i,j) : rem wert einlesen
510 :       if we(i,j)>max then max=we(i,j) : rem maximum
520 :       if we(i,j)<min then min=we(i,j) : rem minimum
530 :   next j : rem weiter machen
540 print "------------------------------------"
```

```
550 next i : rem weitermachen
560 :
570 rem ***********************
580 rem *   achsen zeichen    *
590 rem ***********************
600 :
610 graphic 1,1 : fast : rem grafik an und loeschen; fast
620 :
630 ft=(h3-h1)/600 : rem normalumrechnungsfaktor
640 if max<=600 and min>=-300 then 690 : rem ok ?
650 ft=(h3-h1-1)/abs(max) : rem 1.neuer faktor
660 if h3+abs (min)*ft <=h2 then 690 : rem jetzt ok ?
670 ft=(h2-h3+1)/abs(min) : rem 2.neuer faktor
680 :
690 draw 1,v1,h1 to v2,h1 : rem 1. horizontale
700 draw 1,v1,h2 to v2,h2 : rem 2. horizontale
710 draw 1,v1,h3 to v2,h3 : rem 3. horizontale
720 draw 1,v1,h1 to v1,h2 : rem 1. vertikale
730 draw 1,v2,h1 to v2,h2 : rem 2. vertikale
740 :
750 z=0 : rem index auf null setzen
760 for i=h3 to h1 step -(h3-h1)/6
770 :   x=xl : y=i-3 : t$=str$(z) : gosub1670
780 :   draw 1,v1-2,i to v1+2,i : rem positive y-einteilungen
790 :   z=z+100 : rem wert um 100 erhoehen
800 next i
810 :
820 z=0 : rem index auf null setzen
830 for i=h3 to h2+1 step (h2-h3)/3
840 :   x=xl : y=i-3 : t$=str$(z) : gosub1670
850 :   draw 1,v1-2,i to v1+2,i : rem negative y-einteilungen
860 :   z=z-100 : rem wert um 100 verkleinern
870 next i
880 :
890 for i=1 to 3 : z=1
900 :   for j=v1+sb*1.75 to v2 step sb*3.5
910 :       x=j-4 : y=h2+4+((i-1)*8)
920 :       t$=mid$(mo$(z),i,1) : gosub 1670
930 :       draw 1,j,h2+2 to j,h2-2 : rem x-einteilungen
940 :       z=z+1 : rem index erhoehen
```

```
950 :   next j
960 next i
970 :
980 slow : rem wieder slow-modus aktivieren
990 :
1000 rem ************************
1010 rem *   werte eintragen    *
1020 rem ************************
1030 :
1040 for i=ar to 1 step -1 : rem reihen schleife
1050 :   z=1 : rem indexzaehler auf eins
1060 :   for x=v1-sb to v2-sb*3.5 step sb*3.5
1070 :
1080 :       if we(i,z)>0 then begin : rem positiv ?
1090 :              xr=x+sb :     rem koordinaten
1100 :              zr=h3 :       rem fuer
1110 :              yr=ts*(i-1) : rem positiven
1120 :              x1=fnx(0) :   rem wert berechnen
1130 :              y1=fny(0)
1140 :
1150 :              xr=x+sb*2
1160 :              zr=h3-we(i,z)*ft
1170 :              yr=ts*(i-1)
1180 :              x2=fnx(0)
1190 :              y2=fny(0)
1200 :
1210 :              xr=x+sb
1220 :              zr=h3-we(i,z)*ft
1230 :              yr=ts*i
1240 :              x3=fnx(0)
1250 :              y3=fny(0)
1260 :       bend
1270 :       if we(i,z)<0 then begin : rem negativ ?
1280 :              xr=x+sb :            rem koordinaten
1290 :              zr=h3-we(i,z)*ft :   rem fuer
1300 :              yr=ts*(i-1) :        rem negativen
1310 :              x1=fnx(0) :          rem wert
1320 :              y1=fny(0) :          rem berechnen
1330 :
1340 :              xr=x+sb*2
```

```
1350 :                 zr=h3
1360 :                 yr=ts*(i-1)
1370 :                 x2=fnx(0)
1380 :                 y2=fny(0)
1390 :
1400 :                 xr=x+sb
1410 :                 zr=h3
1420 :                 yr=ts*i
1430 :                 x3=fnx(0)
1440 :                 y3=fny(0)
1450 :         bend
1460 :
1470 :         we(i,z)=abs(we(i,z)) : rem wert ohne vorzeichen
1480 :
1490 :         box 0,x1,y1,x2,y2,0,1 : rem saeule loschen
1500 :         box 0,x2,y3+we(i,z)*ft,x3+sb,y3,0,1
1510 :         for l=0 to x3-x1
1520 :             draw 0,x1+l,y2 to x3+l,y3
1530 :         next l
1540 :
1550 :         box 1,x1,y1,x2,y2 : rem box zeichnen
1560 :         draw 1,x1,y2 to x3,y3 : rem alle raumlinien
1570 :         draw 1,x3,y3 to x3+sb,y3 : rem einzeichnen
1580 :         draw 1,x3+sb,y3 to x2,y2
1590 :         draw 1,x2,y1 to x3+sb,y3+we(i,z)*ft
1600 :         draw 1,x3+sb,y3+we(i,z)*ft to x3+sb,y3
1610 :         z=z+1 : rem index erhoehen
1620 :    next x
1630 next i
1640 :
1650 goto 1650 : rem programmende
1660 :
1670 rem ******************************
1680 rem *   text (x,y,t$) ausgeben   *
1690 rem ******************************
1700 :
1710 xa=(xl and 248)+8 : ya=(yl and 248)+8
1720 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
1730 char 1,xa/8,ya/8,t$
1740 sshape z$,xa,ya,xa+len (t$)*8-1,ya+7
```

```
1750 gshape q$,xa,ya
1760 gshape z$,x,y
1770 return
```

Programmmbeschreibung:

100 - 130:	Programmkopf.
140 - 320:	Definierung aller Konstanten, wie z. B. der Achsenkoordinaten und der Umrechnungsfaktoren.
330 - 350:	Setzen der Farben und Löschen des Textbildschirms.
360 - 550:	Einlesen der Werte.
560 - 590:	Unterprogrammkopf.
600 / 610:	Grafik- und Fastmodus aktivieren.
620 - 670:	Umrechnungsfaktor für die Y-Koordinate berechnen.
680 - 980:	Koordinatensystem, Einteilungen und Monatsabkürzungen einzeichnen.
990 - 1020:	Unterprogrammkopf.
1030 - 1260:	Koordinaten bei positivem Wert berechnen.
1270 - 1450:	Koordinaten bei negativem Wert berechnen.
1460 - 1530:	Raum unter der Säule löschen.
1540 - 1630:	Säule einzeichnen.
1640 / 1650:	Programmende.
1660 - 1770:	Allgemeine Textausgaberoutine.

5.2.3 Die Kuchengrafik

Nachdem Sie nun schon die zwei Arten der Balkengrafik kennengelernt haben, wollen wir Ihnen an dieser Stelle die Kuchendiagramme vorstellen. Wir unterscheiden zwei Arten der Kuchendiagramme:

a) Normale Kuchengrafiken
b) Explosions-Kuchengrafiken

In diesem Kapitel wollen wir uns nun mit der ersten Form beschäftigen, die genau wie auch die zweite Form uns, den Programmierern, großen Kummer bereitet, da nämlich bedauerlicherweise im BASIC 7.0 kein Befehl existiert, mit dem Radien an bestimmten Winkelpositionen angetragen werden können. Doch mit etwas erhöhtem mathematischen Aufwand können wir auch dieses unserem Computer beibringen, mehr dazu weiter unten.

Doch nun erst einmal zur Aufgabe unseres Programms:
Die Aufgabe unseres Programms besteht darin, bestimmte prozentuale Verhältnisse durch verschieden große Kreisstücke auszudrücken. Die Größe der Kreisstücke richtet sich dabei nach dem jeweils darzustellenden Wert und der Gesamtsumme der Werte, ist also direkt proportional. Gäben Sie beispielsweise fünf gleich große Werte an, so würden auch alle fünf Kreisabschnitte gleich groß sein, nämlich jedesmal 72 (360/5) Winkelgrade.

Nun zur Bedienung des Programms:

Das Programm erlaubt es, maximal 20 verschiedene Werte oder andere Angaben einzugeben und diesen jeweils einen Namen zuzuordnen. Bei der Eingabe der Werte ist es für den Anwender nicht notwendig, darauf zu achten, daß alle Werte als Summe den Wert 100% bilden, da das Programm diese Angleichung automatisch vornimmt. Hierbei kann es allerdings zu kleinen Rundungsfehlern aufgrund des Fehlens der ROUND-Funktion kommen, da wir mittels des INT-Befehls nur die Nachkommastellen abschneiden können.

Sollten Sie auch einen Namen mit angegeben haben, so wird dieser auf eine Länge von drei Zeichen gekürzt, doch können Sie auch längere Namen eingeben, wenn Sie die Zeile 380 entfallen lassen. Ist der Fall eingetreten, daß sich zwei Namen überschneiden, so kann man dies durch eine bessere Anordnung der großen und kleinen Werte ändern. Zu den Texten fügt das Programm weiterhin noch den realen, gerundeten Prozentanteil hinzu, wodurch das Ablesen der einzelnen Werte erheblich vereinfacht wird.

Hier nun das Programmlisting zur Erzeugung einer Kuchengrafik:

```
100 rem ********************
110 rem *  kuchengrafik   *
120 rem ********************
130 :
140 xl=  0 : xh=319 : rem x-grenzwerte
150 yl=  0 : yh=199 : rem y-grenzwerte
160 :
170 mx=xl+(xh-xl)/2 : rem x-mittelpunkt
180 my=yl+(yh-yl)/2 : rem y-mittelpunkt
190 rx=(yh-yl)/3 :    rem x-radius
200 ry=rx/3 :         rem y-radius
210 :
220 as=20 :           rem abstand text
230 fa=3.1415/180 :   rem umrechnungsfaktor
240 :
250 color 0,1 : color 4,1 : color 5,14 : rem farben setzen
260 color 1,6 : scnclr :   rem bildschirm loeschen
270 :
280 dim we (20),tx$ (20) : rem speicher reservieren
290 :
300 input "ueberschrift    ";ue$ : rem ueberschrift holen
310 input "anzahl werte    ";az% : rem anzahl werte holen
320 :
330 if az%>20  or az%<1 then 310 : rem erlaubt ?
340 :
350 for i=1 to az%
360 :   print i;". name / wert";
370 :   input tx$ (i),we (i) : rem text, wert einlesen
380 :   tx$ (i)=left$ (tx$ (i)+"   ",3) : rem formatieren
390 :   su=su+we (i) : rem gesamtsumme bilden
400 next i
410 :
420 gs=360/su : rem umrechnungsfaktor
430 :
440 rem ************************
450 rem *   grafik zeichnen    *
460 rem ************************
```

```
470 :
480 graphic 1,1 : rem grafik aktivieren
490 :
500 char 1,((xh-xl)/2-len (ue$)*4)/8+1,yl/8,ue$
510 :
520 circle 1,mx,my,rx,ry : rem ellipse
530 circle 1,mx,my+15,rx,ry,90,270 : rem ellipsenbogen
540 :
550 draw 1,mx-rx,my to mx-rx,my+15 : rem bogen und ellipse
560 draw 1,mx+rx,my to mx+rx,my+15 : rem zusammenfuegen
570 :
580 for i=1 to az%
590 :
600 :    w =(270+(lw+we (i))*gs)*fa   : rem winkel radius
610 :    wd=(270+(lw+we (i)/2)*gs)*fa : rem winkel text
620 :
630 :    x=rx*cos (w)+mx : rem x-rand fuer radius
640 :    y=ry*sin (w)+my : rem y-rand fuer radius
650 :
660 :    draw 1,mx,my to x,y : rem radius einzeichnen
670 :    a=(lw+we (i))*gs : rem formel in 'a' speichern
680 :    if a<90 or a>270 then 710 : rem vorderansicht ?
690 :    draw 1,x,y to x,y+14 : rem ja ! dann zeichnen
700 :
710 :    x=(rx+as/2)*cos (wd)+mx : rem x-text
720 :    y=(ry+as/2)*sin (wd)+my : rem y-text
730 :
740 :    t$=str$ (int (we (i)/(su/100))) : rem prozent
750 :    t$=right$ (t$,len (t$)-1) : rem nur zahl
760 :    tx$ (i)=tx$ (i)+" "+t$+"%" : rem % anhaengen
770 :
780 :    if x<mx then x=x-len (tx$ (i))*8 : rem links ?
790 :    if y>my then y=y+10 : rem rechts vom mittelpunkt
800 :
810 :    if x<xl or x>xh or y<yl or y>yh then 830 : rem ok ?
820 :    t$=tx$ (i) : gosub 890 : rem text ausgeben
830 :
840 :    lw=lw+we (i) : rem wert aufaddieren
850 next i
860 :
```

```
870 goto 870 : rem programmende
880 :
890 rem ******************************
900 rem *   text an x/y ausgeben    *
910 rem ******************************
920 :
930 xa=(xl and 248)+8 : ya=(yl and 248)+8
940 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
950 char 1,xa/8,ya/8,t$
960 sshape z$,xa,ya,xa+len (t$)*8-1,ya7
970 gshape q$,xa,ya
980 gshape z$,x,y
990 return
```

Nun zur Funktionsweise des Programms:

Wie auch alle anderen Programme läuft dieses Programm auf verschiedenen Bildschirmgrößen. Es kann also auch für andere Computer oder den 80-Zeichenmodus genutzt werden, vorausgesetzt, man gleicht die einzelnen Befehle an.

Diese Angleichung sollte aber kein Problem darstellen, da (fast) jede Programmzeile einzeln mit REM dokumentiert und in der Programmbeschreibung aufgeführt ist. Hier jetzt erst einmal die Funktionsbeschreibung und die genauere Erklärung der wichtigsten Programmteile. (Die allgemeine Programmbeschreibung steht weiter unten, in Form einer Tabelle.)

Zuerst müssen wir nun die Grundlagen der Kreisprogrammierung klären. Wie Sie wissen, geschieht in den verschiedenen Ausbaustufen der Kreisgrafik nichts weiter, als daß ein Kreis durch mehrere an verschiedenen Winkeln angetragene Radien in eine Anzahl Teilstücke zerlegt wird. Je nach Ausbaustufe des Programms werden diese Kreisabschnitte dann noch weiter bearbeitet, z. B. ausgefüllt, hervorgehoben, gefärbt, schraffiert usw.

Um nun die eingegebenen Werte in Gradangaben umzurechnen benötigt man einen oder mehrere Umrechnungsfaktoren. Wir wollen aus Geschwindigkeitsgründen diese Umrechnung mit nur einem Angleichfaktor vornehmen, in dem alle anderen Umrech-

nungen gleich enthalten sind. Bei der Bildung dieses Umrechnungsfaktors müssen wir folgendes berücksichtigen:

1. Gesamtsumme der Werte
2. Anzahl der Gradeinteilungen

Die Berechnung der Gesamtsumme, die wir auf jeden Fall benötigen, nehmen wir gleich in der Werteingabeschleife vor, während uns die Anzahl der Gradeinteilungen von vornherein bekannt ist, diese beträgt nämlich 360 Altgrad.

Ist uns nun die Gesamtsumme und die Anzahl der Gradeinteilungen bekannt, so sollten wir folgenden Zusammenhang erkennen können:

360 Grad entsprechen der Gesamtsumme

Um nun herauszubekommen, wieviel Grad dem Wert eins entsprechen, müssen wir die Anzahl der Winkeleinteilungen durch die Gesamtsumme dividieren und jeden Wert dann mit diesem Faktor multiplizieren. Die Formel zur Berechnung dieses Umrechnungsfaktors lautet daher jetzt:

Faktor = 360 / Gesamtsumme

(Im Programm steht diese Umrechnung in der Zeile 420, während die Bildung der Gesamtsumme in der Zeile 390 vorgenommen wird.)

Ein weiteres Problem bei den verschiedenen Formen der Kuchengrafiken stellt, wie oben schon erwähnt, die Zeichnung des Radius an einer bestimmten Winkelposition dar, weil für diese Aufgabe im BASIC 7.0 kein Grafikbefehl implementiert ist. Erinnern wir uns aber an das in Kapitel 4.2 beschriebene Polarkoordinatensystem, so stellen wir fest, daß die beiden dort geforderten Angaben vorhanden sind. Es sind dies die Entfernung vom Koordinatenursprung (0/0) zum Punkt P und der Winkel W, der vom positiven Teil der X-Achse im Gegenuhrzeigersinn abgetragen wird. Wenden wir nun die in diesem Kapitel angegebenen Formeln zur Umrechnung der Polarkoordinaten in Flächen-

koordinaten an, so erhalten wir die Koordinaten des Endpunktes des Radius mit dem Winkel W. Wir brauchen dann nur noch die Koordinaten des Kreismittelpunkts zuaddieren und schon haben wir die Start- und Endkoordinaten unseres Radius. Hier noch einmal die beiden Formeln zur Umrechnung, damit Sie nicht ewig hin- und herblättern müssen:

X = COS (W) * L + MX
Y = SIN (W) * L + MY

X = Flächenkoordinate X
Y = Flächenkoordinate Y
W = Winkel
L = Entfernung zum Punkt P
MX/MY = Koordinaten Kreismittelpunkt

Für den Fall, daß auch Sie einmal einen Befehl benötigen sollten, mit dem Sie einen Radius einzeichnen können, haben wir hier die Routine noch einmal aufgeführt. Sie kann auch zum Einzeichnen des Radius in eine Ellipse dienen. Die beiden Radien die in den Variablen RX und RY enthalten sind, müssen dann nur verschiedene Werte annehmen.

Programm zum Zeichnen eines Radius:

```
10000 rem ******************************
10010 rem * --- radius  zeichnen --- *
10020 rem * mx/my : kreismittelpunkt *
10030 rem * rx/ry : kreisradien      *
10040 rem * w     : winkel           *
10050 rem ******************************
10060 :
10070 fa = 3.1415 / 180
10080 :
10090 xr = rx * cos ((270 + w) * fa) + mx
10100 yr = ry * sin ((270 + w) * fa) + my
10110 :
```

```
10120 draw 1,mx,my to xr,yr
10130 :
10140 return
```

Nachdem wir nun auch das zweite Problem gelöst hätten, wollen wir mit der Funktionsbeschreibung des Kuchendiagramms fortfahren. Ein weiterer wichtiger Programmteil (680/690) ist der Abschnitt des Programms, in dem geprüft wird, ob die Kante der Ellipse sichtbar ist. Falls diese sichtbar ist, so wird eine senkrechte Verbindung zwischen dem oberen und dem unteren Rand gezogen. Dies geschieht einfach, indem wir zur ersten ermittelten Y-Koordinate die Kuchenbreite hinzuaddieren. Diese Simulation eines Kuchens assoziiert beim Betrachter einen dreidimensionalen Eindruck.

Auch die Ausgabe des Textes ist bei weitem nicht so einfach wie bei der Balkengrafik. Daher gehen wir in folgenden Einzelschritten vor:

1. Berechnung der Koordinaten mittels der oben beschriebenen Formeln, wobei wir von verlängerten Radien ausgehen, damit der Text etwas vom Kreis abgehoben wird (710/720).
2. Berechnung des zentrierten Prozentwertes und Anhängen an den normalen Text (730-760).
3. Testen, ob der Text rechts oder links vom Kreismittelpunkt ausgegeben werden soll. Muß er links ausgegeben werden, so wird die Stringlänge von der X-Koordinate subtrahiert (770-790).
4. Überprüfen, ob die Koordinaten erlaubt sind, wenn ja, Ausgabe des Textes (800-820).

Die Textausgaberoutine ist identisch mit denen in den vorhergehenden Programmen.

Tabellarische Programmbeschreibung:

100 - 120:	Programmkopf.
130 - 150:	Festlegung der Bildschirmgrenzkoordinaten.
160 - 180:	Berechnung des Ellipsenmittelpunktes.

190 / 200:	Berechnung der Radien aus den Bildschirmgrenzkoordinaten.
210 / 220:	Abstand des Textes vom Kreisrand.
230:	Umrechnungsfaktor von Radiant in Altgrad.
240 - 260:	Farbsetzung und Löschen des Bildschirms.
270 / 280:	Dimensionierung der Arrays für die Werte und Texte.
290 - 310:	Einlesen von Überschrift und Anzahl der Werte.
320 / 330:	Testen, ob die eingegebenen Werte erlaubt sind, wenn nicht, muß die Eingabe wiederholt werden.
340 - 370:	Einlesen der Werte und der ihnen zugeordneten Texte.
380:	Formatieren des Textes auf drei Zeichen Länge.
390:	Bildung der Gesamtsumme aller eingegebenen Werte.
400:	Schleifenende.
410 / 420:	Berechnung des Umrechnungsfaktors von: Wert nach Grad.
430 - 460:	Unterprogrammkopf.
470 / 480:	Aktivieren und Löschen der HGR.
490 / 500:	Ausgeben der zentrierten Überschrift ohne Bereichsüberprüfung!
510 / 520:	Zeichnen der Ellipse um den Punkt MX/MY mit den Radien RX/RY.
530:	Zeichnen des Bogens um die Koordinaten MX/MY+15 mit den Radien RX/RY.
540 - 560:	Verbinden des Bogens mit dem Kreis, zur Erzeugung des 3-D-Effekts.
570 - 610:	Berechnung des Radius- und des Textwinkels.
620 - 640:	Berechnung der Zielkoordinaten zum Zeichnen des Radius.
650 / 660:	Zeichnen des Radius vom Ellipsenmittelpunkt zum Rand (X/Y).
670 / 680:	Testen, ob sich die Koordinaten auf der Vorderseite des Kuchens befinden.
690:	Ja, dann die beiden Kanten verbinden.
700 - 720:	Berechnung der Flächenkoordinaten des Textes.
730 - 750:	Bildung des prozentualen Anteils und formatieren des Zahlenstrings.
760:	'%' anhängen.

770 / 780: Wenn die Textkoordinaten links vom Kreismittelpunkt liegen, so wird die Textlänge von der X-Koordinate subtrahiert.

790: Ist die Y-Koordinate größer als die des Mittelpunkts, so wird der Text um zehn Punkte nach unten verschoben.

800 - 820: Bereichsüberprüfung und Ausgabe des Textes wenn zulässig.

830 / 840: Aktualisieren des letzten Winkels.

850: Schleifenende.

860 / 870: Programmende.

Die Textausgaberoutine ist identisch mit den vorangegangenen Textausgaberoutinen.

5.2.4 Die Explosions-Kuchengrafik

Die zweite Form der Kuchengrafik, die wir in unserem Statistikkapitel besprechen wollen, ist die Explosions-Kuchengrafik. Vom Prizip her unterscheiden sich die beiden Formen nur wenig, nur die Aufmachung und Behandlung der einzelnen Kreisstücke ist etwas unterschiedlich. So wird bei der Explosionsgrafik kein räumlicher Eindruck durch die Simulation eines Kuchens hervorgehoben, sondern es wird ein bestimmtes Feld hervorgehoben und dieses dann noch gefüllt. Wollte man dieses Statistikprogramm beispielsweise bei irgendeiner Wahl benutzen, so könnte man den Bewerber mit der höchsten Stimmenanzahl hervorheben. Die Bedienung des Programmes ist identisch mit der der normalen Kuchengrafik, nur daß hier eben noch angegeben werden kann, welches Stück hervorgehoben werden soll.

Hier nun wieder das Listing zur Erzeugung einer Explosions-Kuchengrafik:

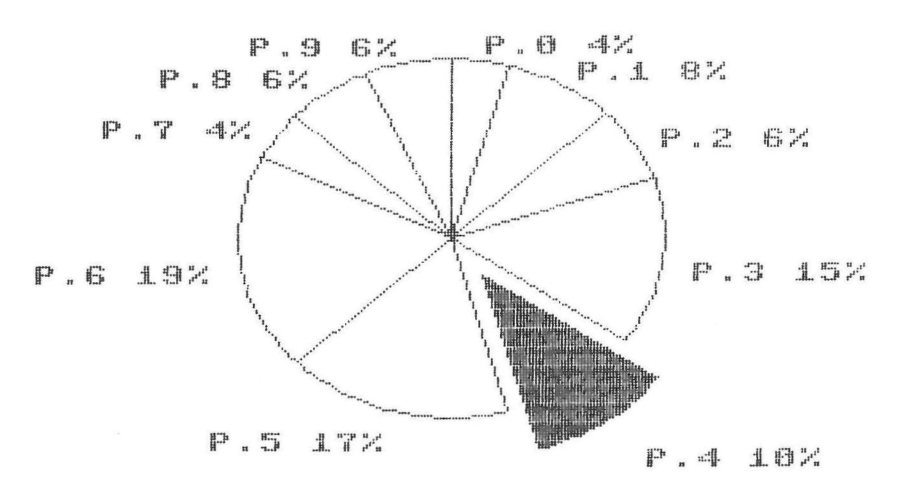
```
100 rem **********************************
110 rem *   explosions-kuchen-grafik    *
120 rem **********************************
130 :
140 xl=  0 : xh=319 : rem x-grenzwerte
150 yl=  0 : yh=199 : rem y-grenzwerte
160 :
170 mx=(xh-xl)/2 : rem x-mittelpunkt
180 my=(yh-yl)/2 : rem y-mittelpunkt
190 :
200 rx=(yh-yl)/2-32 : rem x-radius
210 ry=rx :           rem y-radius
220 :
230 as=rx/4 :         rem abstand text
240 fa=3.14/180 :     rem umrechnungsfaktor
250 :
260 color 0,1 : color 4,1 : color 5,14 : rem farben setzen
270 color 1,6 : scnclr : rem bildschirm loeschen
280 :
290 dim we (20),tx$ (20) : rem speicher reservieren
300 :
310 input "ueberschrift     ";ue$ : rem ueberschrift einlesen
320 input "anzahl werte     ";az% : rem anzahl werte einlesen
330 input "hevorgehoben nr.";hv% : rem nr. explosionsstueck
340 :
350 if az%>20  or az%<0 then 320 : rem erlaubt ?
360 if hv%>az% or hv%<1 then 330 : rem ok ?
370 :
380 for i=1 to az%
390 :   print i;". name / wert";
400 :   input tx$ (i),we (i) : rem name, wert einlesen
405 :   tx$ (i)=left$ (tx$ (i)+"   ",3) : rem formatieren
410 :   su=su+we (i) : rem gesamtsumme bilden
420 next i
430 :
440 gs=360/su : rem umrechnungsfaktor
450 :
460 rem ************************
470 rem *   grafik zeichnen    *
480 rem ************************
```

```
490 :
500 graphic 1,1 : rem grafik aktivieren und loeschen
510 :
520 char 1,((xh-xl)/2-len (ue$)*4)/8+1,yl/8,ue$
530 :
540 for i=1 to az%
550 :
560 :   w1=lw*gs : rem letzter winkel (radiabt)
570 :   w2=(lw+we (i))*gs : rem momentaner winkel (grad)
580 :   w=(270+(w1+(w2-w1)/2))*fa : rem textwinkel (grad)
590 :
600 :   g1=(270+w1)*fa : rem letzter winkel (grad)
610 :   g2=(270+w2)*fa : rem momentaner winkel (grad)
620 :
630 :   if i=hv% then gosub 930 : goto 670 : rem normal ?
640 :
650 :   circle 1,mx,my,rx,ry,w1,w2 : rem kreisbogen w1 -> w2
660 :
670 :   x=(rx-1)*cos (g1)+mx : rem x-koordinate rand (w1/g1)
680 :   y=(ry-1)*sin (g1)+my : rem y-koordinate rand (w1/g1)
690 :
700 :   draw 1,mx,my to x,y : rem radius einzeichnen (w1/g1)
710 :
720 :   x=(rx+as/2)*cos (w)+mx : rem x-koordinate text (w)
730 :   y=(ry+as/2)*sin (w)+my : rem y-koordinate text (w)
740 :
750 :   t$=str$ (int (we (i)/(su/100))) : rem prozent
760 :   t$=right$ (t$,len (t$)-1) : rem nur zahl
770 :   tx$ (i)=tx$ (i)+" "+t$+"%" : rem % anhaengen
780 :   if x<mx then x=x-len (tx$ (i))*8 : rem links ?
790 :   if i<>hv% then 830 : rem explosionsstueck ?
800 :   if x<mx then x=x-as : else : x=x+as : rem lnks/rts
810 :   if y<my then y=y-as : else : y=y+as : rem unten/oben
820 :
830 :   if x<xl or x>xh or y<yl or y>yh then 850 : rem ok ?
840 :   t$=tx$ (i) : gosub 1160 : rem text ausgeben
850 :   lw=lw+we (i) : rem letzter wert
860 next i
870 :
880 goto 880 : rem programmende
```

```
890 :
900 rem ******************************
910 rem * explosionsstueck zeichnen *
920 rem ******************************
930 :
940 xm=as*cos (w)+mx : rem neuer x-kreismittelpunkt
950 ym=as*sin (w)+my : rem neuer y-kreismittelpunkt
960 :
970 circle 1,xm,ym,rx,ry,w1,w2 : rem bogen w1 -> w2
980 :
990 x=(rx-1)*cos (g1)+xm : rem x-koordinate rand (w1/g1)
1000 y=(ry-1)*sin (g1)+ym : rem y-koordinate rand (w1/g1)
1010 :
1020 draw 1,xm,ym to x,y : rem radius zeichnen
1030 :
1040 x=(rx-1)*cos (g2)+xm : rem x-koordinate rand (w2/g2)
1050 y=(ry-1)*sin (g2)+ym : rem y-koordinate rand (w2/g2)
1060 :
1070 draw 1,xm,ym to x,y : rem radius zeichnen (w2/g2)
1080 :
1090 fx=(rx/2+as)*cos (w)+mx : rem x-koordinate flaeche
1100 fy=(ry/2+as)*sin (w)+my : rem y-koordinate flaeche
1110 :
1120 paint 1,fx,fy : rem flaeche fuellen
1130 :
1140 return : rem zurueck zum hauptprogramm
1150 :
1160 rem ******************************
1170 rem *   text an x/y ausgeben    *
1180 rem ******************************
1190 :
1200 xa=(xl and 248)+8 : ya=(yl and 248)+8
1210 sshape q$,xa,ya,xa+len (t$)*8,ya+8
1220 char 1,xa/8,ya/8,t$
1230 sshape z$,xa,ya,xa+len (t$)*8,ya+8
1240 gshape q$,xa,ya
1250 gshape z$,x,y
1260 return
```

Funktions- und Prinzipsbeschreibung:

Die Explosions-Kuchengrafik ist in diesem Buch das Programm, das den schwierigsten mathematischen Aufwand benötigt. Deshalb kommen wir nicht umhin, an dieser Stelle erst einmal ein Schaubild abzudrucken, mittels dessen wir dann im Laufe dieser Erklärung die mathematischen Grundlagen erarbeiten wollen.



Wie Sie der oben stehenden Zeichnung entnehmen können, stellt die Berechnung des hervorgehobenen Stücks das größte Problem dar, so daß wir zuerst mit der Zeichnung der normalen Stücke beginnen wollen und dann erst den Spezialfall behandeln wollen. Wie auch bei der normalen Kuchengrafik läuft die gesamte Zeichnung der Grafik in einer Schleife ab, deren Laufvariable jeweils den Index für den Text und den Wertearray darstellt. Nun berechnen wir zwar genauso wie auch in der normalen Kuchengrafik den direkt vom momentanen Wert abhängigen Winkel, aber wir zeichnen nicht diesen, sondern immer den vorhergehenden Winkel. Dies stellt die günstigste Lösung dar, da wir bei dem Explosionsstück sonst ja noch viel mehr Linien berechnen müßten, um auch unser Explosionsstück abzuschließen. Theoretisch hätten wir in der normalen Kuchengrafik genauso vorgehen können. Nun zur Zeichnung des Explosionsstückes, da ja der Rest des Programms noch bekannt sein dürfte und daher auch verstanden worden sein sollte:

Wenn wir das Explosionsstück zeichnen wollen, so müssen wir das hervorzuhebende Stück einfach auf der Winkelhalbierenden (wh), die den Winkel zwischen W1 und W2 halbiert, um ein bestimmtes Stück nach außen verschieben. Doch wie läßt sich dies bewerkstelligen? Nun, wir gehen einfach wieder vom Polarkoordinatensystem aus und stellen uns vor, daß alle möglichen Kreismittelpunkte der Explosionsstücke auf einem Kreis mit dem Abstand AS liegen. Mittels des Polarkoordinatensystems können wir daher die Koordinaten des neuen Kreismittelpunktes bestimmen, da ja der Abstand vom Koordinatenursprung (hier: Kreismittelpunkt) und der Winkel (hier: Winkelhalbierende) bekannt sind. Wir rechnen also den neuen Kreismittelpunkt nach folgender Formel aus:

```
WH = W1 + (W2 - W1) / 2
XM = MX + (AS + RX) * COS (WH)
YM = MY + (AS + RY) * SIN (WH)
```

WH	= Winkelhalbierende
W1	= Letzter Winkel
W2	= Momentaner Winkel
AS	= Abstand vom Kreismittelpunkt (MX/MY)
XM/YM	= Neuer Kreismittelpunkt
MX/MY	= Alter Kreismittelpunkt

Um den so erhaltenen, neuen Kreismittelpunkt zeichnen wir dann den Bogen von W1 nach W2 ein und füllen die Fläche aus, indem wir wieder mittels des Polarkoordinatensystems einen Punkt innerhalb der Fläche berechnen. Danach bleibt nichts weiter zu tun, als mit RETURN wieder zum Hauptprogramm zurückzuspringen. Natürlich muß im Programm auch noch berücksichtigt werden, daß das Explosionsstück etwas weiter herausragt. Diese Angleichung geschieht in den Zeilen 790 bis 810.

Nun zur allgemeinen Programmbeschreibung, aus der dies eindeutig hervorgeht:

100 - 120:	Programmkopf.
130 - 150:	Festlegen der Bildschirmgrenzkoordinaten.
160 - 180:	Berechnen des Kreismittelpunktes aus den Bildschirmgrenzkoordinaten.
190 - 210:	Berechnen der Radien.
220 / 230:	Abstand des Textes vom Kreisrand und Abstand des Explosionsstückes vom alten Kreismittelpunkt (MX/MY).
240:	Umrechnungsfaktor von Radiant in Grad.
250 - 270:	Setzen der Farben und Löschen des Textschirms.
280 / 290:	Dimensionierung der Arrays für die Texte und die Werte.
300 - 330:	Einlesen der Überschrift, der Anzahl der Werte und der Nummer des Stückes, das hervorgehoben werden soll.
340 - 360:	Überprüfung, ob die eingegeben Werte alle erlaubt sind.
370 - 400:	Einlesen der Namen und der Texte, die bei Bedarf noch formatiert werden. Soll diese Formatierung entfallen, so muß die Zeile 405 wegfallen.
410:	Bildung der Gesamtsumme.
420:	Schleifenende.
430 / 440:	Berechnen des Umrechnungsfaktors aus der Gesamtsumme und der Anzahl der Winkelpositionen.
450 - 480:	Unterprogrammkopf.
490 - 500:	Grafik aktivieren und Löschen.
510 / 520:	Ausgeben der zentrierten Überschrift, allerdings ohne bereichsüberprüfung!!!
530 - 560:	Berechnung des letzten Winkels in Radiant.
570:	Berechnung des momentanen Winkels in Radiant.
580:	Berechnung der Winkelhalbierenden, die den Winkel zwischen W1 und W2 halbiert.
590 / 600:	Berechnung des letzten Winkels, allerdings jetzt in Altgrad.
610:	Berechnung des momentanen Winkels in Altgrad.
620 / 630:	Testen ob das momentan zu zeichnende Stück das zu explodierende ist. Wenn ja, so wird in die Unterroutine ab 900 gesprungen und danach in Zeile 670 fortgefahren.
640 - 650:	Zeichnen des Kreisbogens von W1 nach W2.

660 - 680: Berechnung der Randkoordinaten zum Zeichnen des Radius am Winkel W1.
690 - 700: Zeichnen des Radius.
710 - 730: Berechnen der Textkoordinaten. Winkel W2.
740 / 750: Prozentualen Anteil berechnen und in String speichern.
760: Abschneiden des Blanks am Anfang des Zahlenstrings.
770: Anhängen von '%' an den Zahlenstring.
780: Testen ob die X-Koordinate des Textes links vom Kreismittelpunkt liegt, wenn ja, dann wird die Länge des Zahlenstrings von der X-Koordinate subtrahiert.
790: Wenn das momentan bearbeitete Stück nicht das hervorzuhebende ist, dann wird die folgende Bereichsanpassung übersprungen.
800: Je nachdem, ob der Text links oder rechts vom Kreismittelpunkt liegt, wird die Strecke, um die das Explosionsstück verschoben wurde (AS), von der X-Koordinate abgezogen oder zu dieser hinzuaddiert.
810: Soll der Text unterhalb der Y-Mittelpunktkoordinate ausgegeben werden, so wird die Strecke AS hinzuaddiert, anderenfalls abgezogen.
820 - 840: Ergibt die Bereichsüberprüfung in Zeile 830 ein positives Ergebnis, liegen also die Textkoordinaten innerhalb des erlaubten Bereichs, so wird der Text ausgegeben, falls das Ergebnis nicht positiv ausfällt, wird diese Ausgabe unterlassen.
850: Angleichen des letzten Wertes.
860: Schleifenende.
870 / 880: Programmende.
890 - 920: Unterprogrammkopf.
930 - 950: Ermitteln des neuen Kreismittelpunktes, um den der Kreisbogen des Explosionsstücks gezogen werden soll.
960 / 970: Zeichnen des Kreisbogens von W1 nach W2.
980 - 1000: Ermitteln der Randkoordinate des Explosionsstücks für den Winkel W1.
1010 / 1020: Einzeichnen des Radius am Winkel W1.

1030 - 1060:	Ermitteln der Randkoordinaten für den Winkel W2.
1060 / 1070:	Einzeichnen des Radius am Winkel W2.
1080 - 1100:	Ermitteln eines Punktes innerhalb des Explosionsstücks.
1110 / 1120:	Ausfüllen des Explosionsstücks.
1130 / 1140:	Rücksprung in das Hauptprogramm.
1150 - 1260:	Textausgaberoutine wie auch in allen anderen Programmen. Daher ist die Routine an dieser Stelle nicht mehr dokumentiert.

5.2.5 Gewinn-Verlust-Grafiken (Vergleichsgrafiken)

Wollen Sie verschiedene Wertereihen miteinander vergleichen, so eignet sich weder die Balkengrafik noch eine Form der Kuchengrafik für diesen Zweck. Aus diesem Grund wollen wir Ihnen hier die Vergleichsgrafik vorstellen. Eine Vergleichsgrafik wird aus zwei Kurven gebildet, deren Schnittflächen dann jeweils nach Kurvenverlauf gefüllt werden oder nicht.

Anwenden können Sie diese Form der grafischen Statistik beispielsweise, wenn Sie Ihre Einnahmen mit Ihren Ausgaben vergleichen möchten. Für die Einnahmen nehmen Sie die erste Kurve, während die zweite Kurve dann die Ausgaben darstellt. Sind die Ausgaben höher als die Einnahmen, kommt es also zu einer Überschneidung, so wird der Differenzbereich gefärbt und signalisiert Ihnen, daß Sie in den betreffenden Monaten einen Verlust erwirtschaftet haben. Sie sehen also, für jedes Anwendungsgebiet gibt es eine speziell geeignete Form der Statistik. Hier nun das Programmlisting, an das sich die Programmerklärung anschließt.

Programmlisting:

```
100 rem *************************
110 rem *   vergleichsgrafik    *
120 rem *************************
130 :
140 xl=  0 : xh=319 : rem x-grenzwerte
150 yl=  0 : yh=190 : rem y-grenzwerte
160 :
170 tt=yh+2 : ax=10 : rem y-text,
180 :
190 scnclr : rem bildschirm loeschen
200 :
210 color 0,6 : color 1,1 : color 5,1 : rem farben
220 :
230 input "rastereint. j/n";ra$ : rem raster ?
240 input "12 monate (j/n)";mo$ : rem monate ?
250 ra$=left$ (ra$+"n",1) : rem 'j' oder 'n'
260 :
270 mo$=left$ (mo$+"n",1) : rem 'j' oder 'n'
280 if mo$="j" then aw=12 : goto 330 : rem ja, monate
290 :
300 input "wieviel werte pro kurve";aw : rem anzahl werte
310 if aw>(xh-xl)/3 or aw=0 then 300 : rem erlaubt ?
320 :
330 dim we (aw+1,2) : rem array dimensionieren
340 :
350 for i=1 to aw
360 :   print "wert nr."i;
370 :   input we(i,1),we(i,2) : rem werte einlesen
380 :   a=abs (we(i,1)) : if a>hi then hi=a : rem maximum ?
390 :   a=abs (we(i,2)) : if a>hi then hi=a : rem maximum
400 next
410 :
420 we (i,1)=we (i-1,1) : rem 13. wert = 12. wert
430 we (i,2)=we (i-1,2) : rem 13. wert = 12. wert
440 :
450 yf=1 : a=(yh-yl)/2 : rem y-zerrfaktor = 1
460 if hi>a then yf=a/hi : rem ueberschreitung dann anpassen
470 :
```

```
480 rem ************************
490 rem *   grafik zeichnen    *
500 rem ************************
510 :
520 :
530 graphic 1,1 : rem hires aktivieren und loeschen
540 :
550 for x=xl to xh step (xh-xl)/aw
560 :
570 :   z=z+1 : rem indexzeiger erhoehen
580 :
590 :   w1=yl+(yh-yl)/2-we (z,1)*yf : rem wert 1
600 :   w2=yl+(yh-yl)/2-we (z,2)*yf : rem wert 2
610 :   my=w1+(w2-w1)/2 : di=w1-w2 : rem mittelw., differenz
620 :
630 :   if z<>1 then 660 : rem 1. wert
640 :      x0=x : y0=w1 : y1=w2 : rem ja !
650 :      draw 1,x0,y0 to x0,y1 : rem dann abschliesen
660 :   draw 1,x0,y0 to x,w1 : rem geraden zeichnen
670 :   draw 1,x0,y1 to x,w2
680 :
690 :   if w2<=w1 and fl=0 then fl=1 : xa=x : max=my : dv=di
700 :   if di>dv and fl=1 then xa=x : max=my : dv=di
710 :   if sgn (di)=1 then 760
720 :   if fl=0 then 760
730 :
740 :      paint 1,xa,max : fl=0 : rem flaeche fuellen
750 :
760 :   x0=x : y0=w1 : y1=w2 : rem letzten werte
770 next x
780 :
790 draw 1,x0,y0 to xh,y0 : rem fortfuehren
800 draw 1,x0,y1 to xh,y1 : rem und dann
810 draw 1,xh,y0 to xh,y1 : rem abschliesen
820 :
830 if fl=0 then 880 : rem letzte flaeche testen
840 paint 1,xa,max : rem ja ! auch fuellen
850 :
860 if ra$="n" then 1090 : rem raster ?
870 :
```

```
880 restore : fast : rem ja ! raster
890 :
900 for i=xl to xh step (xh-xl)/aw
910 :   draw 1,i,yl to i,yh : rem senkrechtes raster
920 :   if mo$="n" then 960 : rem monate antragen ?
930 :   read t$ : rem ja
940 :    x=i+2 : y=tt
950 :    gosub 1140
960 next : slow
970 :
980 y=yl+(yh-yl)/2
990 :
1000 draw 1,xl,y-1 to xh,y-1 : rem mittelbalken
1010 draw 1,xl,y+1 to xh,y+1 : rem dicker
1020 draw 1,xl,y to xh,y :     rem zeichnen
1030 :
1040 for i=0 to y-yl-ax step ax
1050 :   draw 1,xl,y+ax+i to xh,y+ax+i : rem horizontales
1060 :   draw 1,xl,y-ax-i to xh,y-ax-i : rem raster
1070 next i
1080 :
1090 goto 1090 : rem programmende
1100 :
1110 data jan,feb,mrz,apr,mai,jun
1120 data jul,aug,sep,okt,nov,dez
1130 data 0 : rem endmarkierung
1135 :
1140 rem **********************
1150 rem *   text ausgeben    *
1160 rem **********************
1170 :
1180 xa=(xl and 248)+8 : ya=(yl and 248)+8
1190 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
1200 char 1,xa/8,ya/8,t$
1210 sshape z$,xa,ya,xa+len (t$)*8-1,ya+7
1220 gshape q$,xa,ya
1230 gshape z$,x,y
1240 return
```

Nun zum Prinzip der Vergleichsgrafik. Wie Sie wahrscheinlich schon richtig gefolgert haben, werden die beiden Kurven auf ähnliche Art und Weise gezeichnet, wie auch in der normalen Kurvengrafik. Der einzige Unterschied zwischen beiden besteht darin, daß in der Vergleichsgrafik noch getestet wird, ob der Wert der zweiten Kurve größer als der der ersten ist, also ein Verlust aufgetreten ist. Ist dies der Fall, so wird ein Flag gesetzt und bei der nächsten Überschneidung der beiden Kurven die entstandene Schnittfläche gefüllt.

Soviel nun zum Funktionsprinzip. Sollten Sie ein richtiger Freak sein, so werden Sie wahrscheinlich eine Programmdokumentation der Erklärung des Prinzips vorziehen, deshalb wie immer auch hier die Programmbeschreibung:

Programmbeschreibung:

100 - 120:	Programmkopf.
130 - 150:	Definition der Bildschirmgrenzen.
160 / 170:	Y-Koordinate des Textes und Abstand der einzelnen Rastereinteilungen zueinander bestimmen.
180 / 190:	Textbildschirm löschen.
200 / 210:	Farben definieren.
220 - 240:	Abfragen, ob ein Raster eingezeichnet werden soll und ob die Monatsabkürzungen am unteren Bildschirmrand ausgegeben werden sollen. Sollen die Monatsabkürzungen ausgegeben werden, so geht das Programm davon aus, daß auch zwölf Werte eingelesen werden sollen.
250 - 280:	Formatieren der einzelnen Strings und Festsetzen der Werteanzahl auf zwölf falls die Frage nach den Monatsabkürzungen mit JA beantwortet wurde.
290 - 310:	Einlesen der Werteanzahl und Bereichsüberprüfung, falls weiter oben NEIN eingegeben wurde.
320 / 330:	Dimensionieren des Arrays, der die einzelnen Werte der beiden Kurven enthalten soll.
340 - 400:	Einlesen der gesamten Werte und Berechnung des Minimums und des Maximums.

410 - 430: In diesen Zeilen wird festgelegt, daß der dem letzten Wert folgende Wert den gleichen Inhalt besitzt wie der letzte.

440 - 460: Berechnung des Umrechnungsfaktors für die Anpassung der Werte an den zur Verfügung stehenden Y-Bereich.

470 - 500: Unterprogrammkopf.

510 - 530: Aktivieren und Löschen der Grafik.

540 - 570: Erhöhen des Array-Zeigers und Schleifenbeginn für die X-Koordinaten.

580 - 610: Berechnung der beiden Werte mit Anpassung an den zur Verfügung stehenden Y-Bereich und Bildung des Mittelwertes dieser beiden Werte. Dieser Mittelwert wird dann eventuell als Y-Koordinate eingesetzt, wenn das Feld gefüllt werden soll.

630 - 650: Sollte es sich um den ersten zu zeichnenden Wert handeln, so werden die beiden Y-Koordinaten durch eine Senkrechte verbunden und die letzten Koordinaten gleich den momentanen Koordinaten gesetzt.

660 / 670: Zeichnung der beiden Linien von den jeweils letzten Koordinaten zu den durch die X-Schleife und durch die umgerechneten Werte gebildeten Koordinaten.

680 / 690: Hier wird getestet, ob der zweite Wert größer als der erste Wert ist. Ist dies der Fall, so wird ein Flag gesetzt und alle wichtigen Werte gespeichert, sofern dies nicht schon vorher geschehen ist (Beachten Sie bitte, daß der Koordinatenursprung in der HI-RES in der linken oberen Ecke liegt).

700: Sollte die Differenz zwischen beiden Werten größer als die vorhergegangene sein, so werden die Maximalwerte mit den momentanen Werten gleichgesetzt.

710 / 720: Sollte der zweite Wert immer noch größer als der erste sein, so wird der Befehl zum Füllen der Differenzfläche übersprungen. Das gleiche geschieht, wenn das Flag nicht gesetzt ist, also kein Verlust aufgetreten war.

730 / 740:	Ausfüllen der Differenzfläche, falls dies notwendig ist und alle weiter oben stehenden Abfragen ein positives Ergebnis lieferten.
750 / 760:	Zwischenspeichern der letzten Werte, die dann bei der Zeichnung der Kurven benutzt werden.
770:	Schleifenende.
780 - 810:	Letztes Feld fortführen bis zur X-Grenzkoordinate und dann beide Werte durch eine Senkrechte verbinden.
820 - 840:	Testen, ob letztes Feld auch noch gefüllt werden muß. Wenn ja, wird dies erledigt.
850 / 860:	Testen, ob ein Rahmen eingezeichnet werden soll. Wenn nicht, so wird der Programmteil bis 1090 übersprungen.
870 / 880:	Aktivieren der FAST-Betriebsart.
890 - 960:	Einzeichnen des vertikalen Rasters und Ausgabe der Monatsabkürzungen, falls dies gewählt wurde.
970 / 980:	Berechnen der Y-Koordinate der horizontalen Nullachse.
990 - 1020:	X-Achse dreimal so dick, wie die anderen Achsen zeichnen.
1030 - 1070:	Einzeichnen des horizontalen Rasters.
1080 / 1090:	Programmende.
1100 - 1130:	Monatsabkürzungen in DATA-Statements abgelegt.
1140 - 1240:	Text an beliebiger Position ausgeben.

5.2.6 Kurvengrafik

Als nächstes im Rahmen unseres Statistikkapitels wollen wir die Kurvengrafik besprechen. Sie eignet sich besonders für große Datenmengen, die dann mit der Kurvengrafik ausgewertet werden können. Also lassen Sie Ihrer Phantasie freien Lauf, denn es gibt Anwendungsmöglichkeit ein. Hier nun das Programm:

Programmlisting:

```
100 rem ********************
110 rem *   kurvengrafik   *
120 rem ********************
130 :
140 xl=  0 : xh=319 : rem x-grenzwerte
150 yl=  0 : yh=190 : rem y-grenzwerte
160 :
170 tt=yh+2 : rem y-koordinate text
180 :
190 scnclr : rem textschirm loeschen
200 :
210 color 0,6 : color 1,1 : color 5,1 : rem farben setzen
220 :
230 input "anzahl der kurven";ak :  rem anzahl kurven
240 :
250 input "rastereint. j/n  ";ra$ : rem raster ja/nein
260 :
270 input "12-monate j/n    ";mo$ : rem monatsabkuerzungen
280 mo$=left$ (mo$+"n",1) :         rem string formatieren
290 :
300 if mo$="j" then aw=12 : goto 360 : ren ja abkuerz.
310 :
320 input "anzahl werte     ";aw : rem anzahl der werte
330 :
340 if aw>(xh-xl)/3 or aw=0 then 320 : rem erlaubt ?
350 :
360 dim we(ak,aw+1) : rem speicher reservieren
370 :
380 for i=1 to ak : rem anzahl kurven aussenschleife
390 :   for j=1 to aw : rem anzahl werte innenschleife
400 :       print "reihe:";i;"wert:";j;
410 :       input we(i,j) : rem wert einlesen
420 :       a=abs(we(i,j)) : rem maximum berechnen
430 :       if a>hi then hi=a : rem und evtl. korrigieren
440 :   next j
450 :
460 :   we (i,j)=we(i,j-1) : rem letzter w. = vorletzter w.
470 :
```

```
480 next i
490 :
500 yf=1 : a=(yh-yl)/2 :   rem y-umrechnungsfaktor
510 if hi>a then yf=a/hi : rem berechnen
520 :
530 rem ************************
540 rem *   grafik zeichnen    *
550 rem ************************
560 :
570 graphic 1,1 : rem grafik aktivieren und loeschen
580 :
590 for i=1 to ak
600 :    z=0 : rem index auf null setzen
610 :    for x=xl to xh step (xh-xl)/aw
620 :        z=z+1 : rem index erhoehen
630 :        y=yl+(yh-yl)/2 :  rem y-koordinate
640 :        y=y-we (i,z)*yf : rem berechnen
650 :
660 :        if z=1 then x0=xl : y0=y : rem erster wert ?
670 :
680 :        draw 1,x0,y0 to x,y : rem linie zeichnen
690 :
700 :        x0=x : y0=y : rem alte koordinaten speichern
710 :    next x
720 :
730 :    t$=str$(i) : x=x0-len (t$)*8 : rem nummer in string
740 :    y=y0-4 : gosub 1040 : rem text ausgeben
750 :
760 next i
770 :
780 if mo$="n" then 850 : rem monatsabkuerzungen
790 :
800 for i=xl to xh step (xh-xl)/aw
810 :    read t$ : rem monatsabkuerzungen ausgeben
820 :    x=i+2 : y=tt : gosub 1040
830 next i
840 :
850 box 1,xl,yl,xh,yh : rem rahmen zeichnen
860 draw 1,xl,yl+(yh-yl)/2 to xh,yl+(yh-yl)/2
870 :
```

```
880 for x=xl to xh step aw
890 :   draw 1,x,yl to x,yl+2 : rem x-einteilungen
900 :   draw 1,x,yh to x,yh-2 : rem zeichnen
910 next x
920 :
930 for y=yl to yh step 10*yf
940 :   draw 1,xl,y to xl+2,y : rem y-einteilungen
950 :   draw 1,xh,y to xh-2,y : rem zeichnen
960 next y
970 :
980 goto 980 : rem programmende
990 :
1000 data jan,feb,mrz,apr,mai,jun
1010 data jul,aug,sep,okt,nov,dez
1020 data 0
1030 :
1040 rem **********************
1050 rem *   text ausgeben    *
1060 rem **********************
1070 xa=(xl and 248)+8 : ya=(yl and 248)+8
1080 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
1090 char 1,xa/8,ya/8,t$
1100 sshape z$,xa,ya,xa+len (t$)*8-1,ya+7
1110 gshape q$,xa,ya
1120 gshape z$,x,y
1130 return
```

Programmbeschreibung:

100 - 120:	Programmkopf.
130 - 150:	Definition der Bildschirmgrenzwerte.
160 / 170:	Y-Koordinate, an der die Monatsabkürzungen ausgegeben werden sollen.
180 - 210:	Bildschirm löschen und Farben setzen.
220 - 340:	Einlesen der Werteanzahl, der Kurvenanzahl usw.
350 / 360:	Dimensionierung des Arrays für die einzelnen Werte.
370 - 480:	Einlesen der Werte und Berechnung des Maximums und des Minimums.

490 - 510:	Berechnung des Y-Umrechnungsfaktors.
520 - 550:	Unterprogrammkopf.
560 / 570:	Aktivieren und Löschen der Grafik.
580 - 760:	Zeichnen der einzelnen Kurven. Die Umrechnung der Werte erfolgt mittels des weiter oben berechneten Umrechnungsfaktor YF.
770 - 830:	Wenn erlaubt, Ausgabe der Monatsabkürzungen.
840 - 960:	Zeichnung des Koordinatensystems.
970 / 980:	Programmende.
990 - 1020:	In DATA-Statements abgelegte Monatsabkürzungen.
1030 - 1130:	Textausgaberoutine.

5.3 Funktionsplotter

5.3.1 2-D-Funktionsplotter

Wollen Sie eine Funktion berechnen, so können Sie dies einerseits in Form einer unübersichtlichen Tabelle oder andererseits in Form einer Grafik vornehmen. Wir wollen nun die zweite Form, also die grafische Darstellung, für Sie wesentlich vereinfachen. Zum Zeichnen einer Funktion benötigen Sie jetzt nämlich nur die folgenden Dinge:

a) Computer + Monitor
b) Die Funktion
c) Unser Grafikbuch
d) Einen Drucker

Zumindest die Punkte a bis c können wir voraussetzen.
Doch was macht nun unser 2-D-Funktionsplotter. Nun, eigentlich ist dies ganz einfach. Wenn Sie ein bißchen Mathematikkentnisse haben, so wird Ihnen der Begriff Funktion sicher bekannt sein. Für die, die keine Mathespezialisten waren oder sind, hier noch einmal die Erklärung dieses Begriffs:

Eine Funktion ist die eindeutige Zuordnung eines Wertes zu einem anderen.

Wir ordnen also einem Wert auf der horizontalen Achse (x) genau einen Wert auf der vertikalen Achse (y) zu.

Doch nun etwas spezieller zu dem Funktionsplotter:

Als erstes ermöglicht der Funktionsplotter natürlich die Eingabe der Funktion. Hierbei können Sie entweder die alte Funktion übernehmen oder eine neue eingeben. Nachdem Sie dann RETURN gedrückt haben, übernimmt der Computer die Funktion in den Programmtext. Hierbei tritt manchmal ein SYNTAX ERROR auf, obwohl die Funktion korrekt eingegeben wurde. Starten Sie in diesem Fall das Programm noch einmal.

Nachdem die Funktion übernommen wurde, verlangt das Programm dann die einzelnen Zerrfaktoren. Wird kein Zerrfaktor eingegeben, so geht das Programm vom Zerrfaktor eins aus. Über den X-Zerrfaktor kann bestimmt werden, ob der X-Wert der in die Funktion eingesetzt wird, größer (Faktor größer als eins), kleiner (Faktor kleiner als eins) oder gleich (Faktor gleich eins) dem in der Schleife hochgezählten X-Wert ist. Der Y-Faktor dient zur Scalierung der Funktion. D.h., mit ihm kann der Graph in Z-Richtung vergrößert (Faktor größer als eins), verkleinert (Faktor kleiner als eins) oder beibehalten (Faktor gleich eins) werden.

Nach dieser Angabe muß der Benutzer die Wertebereiche der Funktion durch ein Komma getrennt eingeben. Der Nullpunkt des Koordinatensystems liegt immer in der Bildschirmmitte, der VON-Wert sollte also nicht größer als null oder dem BIS-Wert sein, da dies zu Störungen im Programmablauf führen kann. Mit welcher Schrittweite die Werte in der X-Schleife berechnet werden sollen, geben Sie auf die Frage nach der X-Schrittweite an.

Als weitere Option ist es noch möglich, den zahlenmäßigen Abstand der Achseneinteilungen zu wählen. Wird die Einteilung für die X-Achse beispielsweise auf zehn gesetzt, so entspricht je-

weils eine Einteilung dem Wert zehn. Analog hierzu funktioniert auch die Angabe der Z-Einteilungen.

Möchten Sie eine alte Grafik überschreiben, so geben Sie auf die Frage, ob die Grafik gelöscht werden soll, bitte ein 'N' für 'NEIN' ein. An dieser Stelle soll nun das Programm folgen, an das sich dann die Programmbeschreibung anschließt.

Programmlisting:

```
100 rem ****************************
110 rem *   2-d-funktionsplotter   *
120 rem ****************************
130 :
140 scnclr : rem bildschirm loeschen
150 :
160 fu$="x*x
170 :
180 print spc(10);fu$ : rem alte funktion ausgeben
190 print chr$ (19); : rem cursor home
200 input "f(x,y)= ";fu$ : rem neue funktion einlesen
210 :
220 color 0,6 : color 1,1 : color 4,7 : rem farben setzen
230 color 5,1 : scnclr : rem bildschirm loeschen
240 :
250 print "160 fu$="+chr$(34)+fu$
260 print "660 def fn y(a)=";fu$
270 print "fu$=";chr$(34);fu$;chr$(34);
280 print ":goto 380"
290 :
300 for i=843 to 845 : rem return's in den
310 :   poke i,13 :     rem tastaturpuffer
320 next i :            rem schreiben
330 :
340 poke 842,19 : poke 208,4 : rem home und anzahl zeichen
350 :
360 end : rem befehle im tastaturpuffer ausfuehren
370 :
380 scnclr : rem textschirm loeschen
```

```
390 :
400 input "y-zerrfaktor       ";zy : rem y-zerrfaktor
410 if zy=0 then zy=1 : rem null ?
420 :
430 input "x-zerrfaktor       ";zx : rem x-zerrfaktor
440 if zx=0 then zx=1 : rem null ?
450 :
460 input "x-bereich (von,bis)";xv,xb : rem x-bereich
470 :
480 input "x-schrittweite     ";xs : rem x-schrittweite
490 :
500 input "y-bereich (von,bis)";yv,yb : rem y-bereich
510 :
520 input "x-einteilungen     ";xe : rem schrittweite x-eint.
530 :
540 input "y-einteilungen     ";ye : rem schrittweite y-eint.
550 :
560 input "grafik loeschen j/n";ls$ : rem grafik loeschen
570 ls$=left$(ls$+"n",1) : rem string formatieren
580 :
590 rem ************************
600 rem *   grafik zeichnen    *
610 rem ************************
620 :
630 xl=  0 : xh=319 : rem x-grenzwerte
640 yl=  0 : yh=199 : rem y-grenzwerte
650 :
660 def fn y(a)=x*x
670 :
680 graphic 1,0 : if ls$="j" then scnclr : rem grafik an
690 :
700 mx=xl+(xh-xl)/2 : rem grafikmittelpunkt x
710 my=yl+(yh-yl)/2 : rem grafikmittelpunkt y
720 :
730 ym=yb : if yv>yb then ym=yv : rem y-maximum (absolut)
740 xm=xb : if xv>xb then xm=xv : rem x-maximum (absolut)
750 :
760 xf=(mx-xl)/xm : rem x-zerrfaktor
770 yf=(my-yl)/ym : rem y-zerrfaktor
780 :
```

```
790 for i=xv to xb step xs
800 :   x=i*zx : rem x-wert fuer funktion zerren
810 :   yk=my-(fny(0)*zy)*yf : rem gezerrte y-koordinate
820 :   xk=mx+i*xf :      rem x-koordinate
830 :   if i=xv then lx=xk : ly=yk : rem erster wert ?
840 :   if xk<0 or yk<0 then 870 : rem auserhalb des
850 :   if lx<0 or ly<0 then 870 : rem fensters ?
860 :   draw 1,lx,ly to xk,yk : rem linie zeichnen
870 :   lx=xk : ly=yk : rem letzten werte speichern
880 next i
890 :
900 rem **************************
910 rem *   koordinatensystem    *
920 rem **************************
930 :
940 box 1,xl,yl,xh,yh : rem rahmen
950 draw 1,xl,my to xh,my : rem x-achse
960 draw 1,mx,yl to mx,yh : rem y-achse
970 :
980 for x=mx to xh step xe*xf
990 :   draw 1,x,yl to x,yl+2 : rem x-einteilungen
1000 :   draw 1,x,yh to x,yh-2
1010 :   draw 1,x-mx,yl to x-mx,yl+2
1020 :   draw 1,x-mx,yh to x-mx,yh-2
1030 :   draw 1,x,my-2 to x,my+2
1040 :   draw 1,x-mx,my-2 to x-mx,my+2
1050 next x
1060 :
1070 for y=my to yh step ye*yf
1080 :   draw 1,xl,y to xl+2,y : rem y-einteilungen
1090 :   draw 1,xh,y to xh-2,y
1100 :   draw 1,xl,y-my to xl+2,y-my
1110 :   draw 1,xh,y-my to xh-2,y-my
1120 :   draw 1,mx-2,y to mx+2,y
1130 :   draw 1,mx-2,y-my to mx+2,y-my
1140 next y
1150 :
1160 goto 1160 : rem programmende
```

Programmbeschreibung:

100 - 120:	Programmkopf.
130 / 140:	Textbildschirm löschen.
150 / 160:	Funktionsstring definieren, für den Fall, daß die alte Funktion übernommen werden soll.
170 - 200:	Funktion einlesen.
210 - 230:	Farben setzen und Textschirm nochmals löschen.
240 - 360:	Neue Funktion mittels des Tastaturpuffers in den Programmtext übernehmen.
370 - 570:	Grenzwerte, Schrittweiten usw. einlesen.
580 - 610:	Unterprogrammkopf.
620 - 660:	Bildschirmgrenzwerte und Funktion definieren.
670 / 680:	Grafik aktivieren und gegebenenfalls auch noch löschen.
690 - 770:	Koordinatenursprung und Zerrfaktoren definieren bzw. berechnen.
780 - 880:	Gezerrte Funktion einzeichnen, indem der jeweils letzte, berechnete Wert mit dem gerade berechneten Wert durch eine Linie verbunden wird. Das Programm geht also davon aus, daß die Funktion auch für alle Werte zwischen dem letzten und dem momentanen X-Wert definiert ist.
890 - 920:	Unterprogrammkopf.
930 - 1160:	Koordinatensystem inclusive der vorgewählten Markierungen einzeichnen.

5.3.2 3-D-Funktionsplotter

Nachdem wir schon in den vorherigen Kapiteln mit Kurven und Geraden allerhand herumhantiert haben, hier nun auch noch ein 3-D-Funktionsplotter, mit dem es auf komfortable Weise möglich ist, dreidimensionale Funktionen zu berechnen. Wie ein dreidimensionales Koordinatemsystem aufgebaut ist, können Sie in den Kapiteln über die Parallelprojektion und über die Zentralprojektion nachlesen. Zusammenfassend an dieser Stelle noch einmal die Bezeichnungen der drei Achsen:

XR: Horizontale Raumachse (Breite)
ZR: Vertikale Raumachse (Höhe)
YR: Raumtiefenachse (Tiefe)

Der Funktionsplotter verlangt nun eine Formel, mit der es für ihn möglich ist die ZR-Koordinate anhand der XR- und der YR-Koordinaten zu berechnen. Diese Formel hat das Format:

f(XR,ZR)= ...

Außer dieser Formel muß das Programm natürlich noch wissen, für welchen Wertebereich es die Funktion berechnen soll. Es müssen also die YR-Grenzwerte, die XR-Grenzwerte und die Größe des Bereiches, in dem die Funktion liegen soll, angegeben werden. Die angegebenen Koordinaten werden so umgerechnet, daß die jeweiligen Maximalwerte immer direkt am Bildschirmrand liegen.
Aufgrund dieser Tatsache kann es vorkommen, daß Teile der Funktion nicht sichtbar sind, was man aber durch die Angabe eines Verkleinerungsfaktors für die jeweilige Achse ändern kann. Doch dazu etwas weiter unten. Außer dem Wertebereich, von dem die Funktion gezeichnet werden soll, verlangt das Programm noch die Schrittweite, mit der in den einzelnen Schleifen vorgegangen werden soll. Diese Angaben müssen nur für die XR- und die YR-Achse angegeben werden, da die ZR-Werte ja berechnet werden.
Dann erwartet das Programm noch, wie oben schon angekündigt die Verkleinerungsfaktoren, durch die die berechneten Werte dividiert werden. Bei der ZR-Achse errechnet sich der resultierende Faktor aus dem Zerrfaktor dividiert durch den Verkleinerungsfaktor.

Nun kommen wir zu den besonderen Leckerbissen unseres 3-D-Funktionsplotters! Gemeint ist das Löschen von optisch verdeckten Linien. Dadurch sieht man immer nur die Linien, die man auch wirklich sehen darf. Das Prinzip dieses Löschens ist folgendes: Wir löschen unterhalb der gezeichneten Linie immer einen bestimmten Bildschirmbereich, wodurch alles, was räumlich tiefer liegt, gelöscht wird. Da Sie nun wissen, wie hier das

Löschen der verdeckten Linien funktioniert, verstehen Sie bestimmt auch die Aufforderung des Programms, die Löschstrecke anzugeben.
Um den räumlichen Aspekt noch zu verstärken bietet das Programm noch ein weiteres Bonbon an. Es ist auch noch möglich, die Grafik zu vernetzen. Dies funktioniert nach folgendem einfachen, aber wirkungsvollen Prinzip:

1. Wir teilen die X-Achse in eine bestimmte Anzahl Teilstücke auf und dimensionieren für diesen Zweck einen Array.

2. Wir zeichnen die erste Kurve und speichern alle X- und Y-Koordinaten in diesem Array ab.

3. Beim Zeichnen der nächsten Kurven verbinden wir die momentanen Koordinaten mit den jeweils im Array gespeicherten parallel liegenden Koordinaten.

Da alle Linien mittels der Parallelprojektion gezeichnet werden, ist es auch möglich, den Sichtwinkel zu verändern. Dieser sollte sich im Bereich von 0 bis 45 Grad bewegen, wobei 45 Grad zu empfehlen wären. Weiterhin bietet das Programm nur noch einige Standardfunktionen, wie z. B. Speichern und Löschen. Diese werden hier nicht weiter erklärt, da sie eigentlich selbstverständlich sein sollten.

Programmlisting:

```
100 rem ******************************
110 rem *   3-d-funktionsplotter   *
120 rem ******************************
130 :
140 scnclr : rem bildschirm loeschen
150 :
160 fu$="sin(.05*sqr(x*x+y*y))
170 :
180 print spc(10);fu$ : rem alte funktion ausgeben
190 print chr$ (19); : rem cursor home
200 input "f(x,y)= ";fu$ : rem neue funktion einlesen
```

```
210 :
220 color 0,6 : color 1,1 : color 4,7 : rem farben setzen
230 color 5,1 : scnclr : rem bildschirm loeschen
240 :
250 print "160 fu$="+chr$(34)+fu$
260 print "870 def fn yr(a)=";fu$
270 print "fu$=";chr$(34);fu$;chr$(34);
280 print ":goto 380"
290 :
300 for i=843 to 845 : rem return's in den
310 :   poke i,13 :     rem tastaturpuffer
320 next i :            rem schreiben
330 :
340 poke 842,19 : poke 208,4 : rem home und anzahl zeichen
350 :
360 end : rem befehle im tastaturpuffer ausfuehren
370 :
380 scnclr : rem bildschirm loeschen
390 :
400 yp=1 : xp=1 : zp=1 : rem verkleinerungsfaktoren
410 :
420 print "f(x,y)=";fu$ : rem neue funktion ausgeben
430 :
440 print : rem line feed
450 :
460 input "yr-bereich (von,bis)";yv,yb
470 :
480 input "xr-bereich (von,bis)";xv,xb
490 :
500 input "zr-bereich (von,bis)";zv,zb
510 :
520 input "yr-schrittweite     ";ys
530 :
540 input "xr-schrittweite     ";xs
550 :
560 input "zr-verzerrung       ";zz
570 :
580 input "yr-verkleinerung    ";yp : rem verkleinerungs
590 :
600 input "xr-verkleinerung    ";xp : rem faktoren
```

```
610 :
620 input "zr-verkleinerung    ";zp : rem einlesen
630 :
640 input "clipping j/n        ";cl$ : rem clipping ?
650 :
660 cl$=left$ (cl$+"n",1) : rem string formatieren
670 if cl$="n" then 710 : rem kein clipping dann 710
680 :
690 input "loeschstrecke       ";ls : rem anzahl reihen
700 :
710 input "yr-sichtwinkel 0-90 ";sw% : rem achsenwinkel
720 :
730 input "grafik loeschen j/n ";er$ : rem loeschflag
740 er$=left$ (er$+"n",1) : rem string formatieren
750 :
760 input "vernetzen j/n       ";vn$ : rem vernetzen ?
770 vn$=left$ (vn$+"n",1) : rem string formatieren
780 :
790 input "grafik abspeichern  ";gs$ : rem speichern ?
800 gs$=left$ (gs$+"n",1) : rem string formatieren
810 if gs$="n" then 840 : rem bei nein nach 840
820 :
830 input "dateiname           ";na$
840 :
850 if sw%<0 or sw%>45 then sw%=45 : rem winkel erlaubt
860 :
870 def fn yr(a)=sin(.05*sqr(x*x+y*y))
880 def fn  x(a)=xr+yr*cos (sw%*fa)+xm
890 def fn  y(a)=zr+yr*sin (sw%*fa)+ym
900 :
910 xm=160 : rem x-mittelpunkt
920 ym=100 : rem y-mittelpunkt
930 :
940 fa= /180 : rem umrechnungsfaktor radiant -> grad
950 :
960 if vn$="j" then dim lw((xb-xv)/xs+2,2)
970 :
980 :rem ************************
990 :rem *   grafik zeichnen    *
1000 rem ************************
```

```
1010 :
1020 zd=abs (zb-zv) : rem differenz zr-grenzwerte
1030 zf=199/zd :      rem zr-zerrfaktor
1040 :
1050 xd=abs (xb-xv) : rem differenz xr-grenzwerte
1060 xf=319/xd :      rem xr-zerrfaktor
1070 :
1080 yd=abs (yb-yv) : rem differenz yr-grenzwerte
1090 h1=(zb*zf)/sin (sw%*fa) : rem laenge hypoth. i
1100 h2=(zv*zf)/cos (sw%*fa) : rem laenge hypoth. ii
1110 hy=abs (h2-h1) :          rem gesamtlaenge hypoth.
1120 yf=hy/yd :                rem yr-zerrfaktor
1130 :
1140 zf=zf*zz : rem zr-faktor
1150 :
1160 graphic 1,0 : rem grafik aktivieren und
1170 :
1180 if er$="j" then scnclr : rem gegebenenfalls loeschen
1190 :
1200 for y=yb to yv step -ys : rem yr-schleife
1210 :   i=0 : rem indexzaehler auf null
1220 :   for x=xv to xb step xs : rem xr-schleife
1230 :       i=i+1 : rem index erhoehen
1240 :
1250 :       xr=(x*xf)/xp : rem xr-raumkoordinate
1260 :       yr=(y*yf)/yp : rem yr-raumkoordinate
1270 :       zr=(fnyr(0)*zf)/zp : rem zr-raumkoordinate
1280 :
1290 :       xx=fnx(0) : yy=199-fny(0) : rem flaechenkoord.
1300 :
1310 :       if x=xv then xl=xx : yl=yy : rem erster wert?
1320 :
1330 :       if xx<0 or xl<0 then 1570 : rem erlaubt ?
1340 :       if yy<0 or yl<0 then 1570 : rem erlaubt ?
1350 :
1360 :       if cl$="n" then 1490 : rem clipping ?
1370 :
1380 :       if yl>yy then y1=yl : y2=yy
1390 :       if yy>yl then y1=yy : y2=yl
1400 :       if y2-y1=0 then 1460 : rem senkrecht
```

```
1410 :
1420 :        for q=0 to abs(yy-yl) : rem dreieck
1430 :            draw 0,xl+1,yl+q to xx,yy+q
1440 :        next q : rem loeschen
1450 :
1460 :        if yl>yy then box 0,xl+1,yl,xx,yl+ls,0,1
1470 :        if yy>yl then box 0,xl+1,yy,xx,yy+ls,0,1
1480 :
1490 :        draw 1,xl,yl to xx,yy : rem linie zeichnen
1500 :
1510 :        if vn$="n" then 1580 : rem vernetzen
1520 :
1530 :        if lw(i,1)<=0 or lw(i,2)<=0 then 1570
1540 :
1550 :        draw 1,lw(i,1),lw(i,2) to xx,yy
1560 :
1570 :        lw(i,1)=xx : lw(i,2)=yy
1580 :        xl=xx : yl=yy : rem werte aktualisieren
1590 :
1600 :    next x : rem ende xr-schleife
1610 next y : rem ende yr-schleife
1620 :
1630 if gs$="j" then bsave (na$),b0,p7168 to p16383
1640 :
1650 getkey a$ : rem auf taste warten
1660 :
1670 graphic 0 : rem grafik aus
1680 :
1690 end : rem programmende
```

Programmbeschreibung:

100 - 120:	Programmkopf.
130 / 140:	Textbildschirm löschen.
150 / 160:	Funktionsstring.
170 - 200:	Alte Funktion ausgeben und neue einlesen oder die alte Funktion übernehmen.
210 - 230:	Farben festlegen und Bildschirm löschen.

240 - 360:	Programmzeilen mittels des Tastaturpuffers in den BASIC-Text übernehmen.
370 / 380:	Textbildschirm löschen.
390 / 400:	Vordefinitionen für Verkleinerungsfaktoren.
410 - 440:	Alte Funktion ausgeben.
450 - 500:	Bereiche für Achsen festlegen.
510 - 540:	Einlesen der Schrittweiten, mit denen die Funktion berechnet werden soll.
550 / 560:	Einlesen der ZR-Verzerrung.
570 - 620:	Abfragen der Verkleinerungsfaktoren für die Grafik.
630 - 670:	Eingabe, ob das Clipping aktiviert sein soll, und Formatieren des Strings.
680 / 690:	Abfrage des Programms, wie weit nach unten gelöscht werden soll.
700 - 710:	Einlesen des Sichtwinkels, in dem die YR-Achse in den Raum ragt.
720 - 740:	Vom Benutzer erfragen, ob die Grafik gelöscht werden soll.
750 - 770:	Soll die Grafik vernetzt werden, so muß diese Abfrage mit JA beantwortet werden.
780 - 830:	Parameter festlegen, ob und wenn ja, unter welchem Namen die Grafik auf der Floppy abgespeichert werden soll.
840 / 850:	Überprüfen, ob der eingegebene Winkel im Bereich von null bis 45 Grad liegt. Liegt er nicht innerhalb dieses Bereiches, so muß die Eingabe wiederholt werden.
860 - 890:	Definieren der Funktion und der Funktionen zur Umrechnung der Parallelkoordinaten in Flächenkoordinaten.
900 / 920:	In dieser Zeile wird die Hälfte der Maximalkoordinaten berechnet.
930 / 940:	Umrechnungsfaktor von Grad in Radiant.
950 / 960:	Wenn die Vernetzung angewählt wurde, wird der nötige Speicher reserviert.
970 - 1000:	Unterprogrammkopf.
1010 - 1020:	Differenz der ZR-Werte zur Berechnung der Reallänge der ZR-Achse.
1030:	Umrechnungsfaktor für die ZR-Werte.

1040 / 1050: Differenz der ZR-Werte zur Berechnung der Reallänge der XR-Achse.
1060: Berechnung der XR-Zerrfaktor.
1070 / 1080: Differenz der YR-Grenzwerte (Raumachse).
1090: Berechnung des positiven Teils der YR-Achse mittels der Winkelsätze im rechtwinkeligen Dreieck.
1100: Berechnung des negativen Teils nach der oben beschriebenen Methode.
1110: Gesamtlänge der YR-Achse.
1120: Berechnung des YR-Zerrfaktors aus den oben ermittelten Werten.
1130 / 1140: ZR-Faktor mit dem eingegeben Zerrfaktor multiplizieren.
1150 / 1160: Grafik aktivieren und nicht löschen.
1170 / 1180: Wenn ausgewählt, dann Grafikbildschirm löschen.
1190 - 1210: YR-Werte vom geringsten (hinten) zum höchsten (vorn) Wert durchgehen. Indexzeiger für Speicherung bei Vernetzung auf null setzen.
1220: Schleifenbeginn für die XR-Werte die vom geringsten (links) zum höchsten (rechts) Wert mit der gewählten Schrittweite durchgegangen werden.
1230: Erhöhen des Index für die Positionsspeicherung, falls auch die Vernetzung ausgewählt wurde.
1240 - 1270: Berechnung der verzerrten Raumkoordinaten.
1280 / 1290: Umrechnung der Raumkoordinaten in die Flächenkoordinaten.
1300 / 1310: Falls es sich um den ersten Wert handelt, so wird der letzte Wert gleich dem momentanen gesetzt. Dies geschieht, weil sonst immer eine Linie vom Koordinatenursprung zum momentanen Punkt gezogen würde.
1320 - 1340: Testen, ob die Flächenkoordinaten zu klein sind. Sollten sie allerdings die obere Bildschirmgrenze überschreiten, so ist dies relativ unwichtig, da das BASIC 7.0 dann die Linie am Bildschirmrand clippt.

1350 / 1360: Überprüfen, ob die verdeckten Linien gelöscht werden sollen. Ist dies nicht der Fall, so wird die folgende Routine übersprungen und zur Zeile 1490 verzweigt.
1370 - 1390: Ermittlung des kleineren ZR-Wertes für das Löschen der verdeckten Linien.
1400: Handelt es sich um eine Senkrechte, so wird kein Dreieck gelöscht.
1410 - 1440: Löschen des Dreiecks.
1450 - 1470: Löschen eines Rechtecks unterhalb der Linie. Das Löschen erfolgt bis zu der durch die Löschstrecke definierten ZR-Koordinate.
1480 / 1490: Erst jetzt den letzten Punkt mit dem gerade berechneten Punkt verbinden.
1500 - 1560: Grafik, wie oben in der Erklärung beschrieben, vernetzen.
1570 - 1590: Aktualisieren der Zwischenspeicher für die letzten Koordinaten.
1600 - 1620: Schleifenenden.
1630 - 1700: Grafik gegebenfalls speichern, auf Taste warten, Grafik ausschalten und dann in den Direktmodus springen.

Anmerkung zum Programm:

Bei der Ermittlung der ZR-Koordinate mittels der FN-Funktion kann manchmal ein SYNTAX ERROR auftreten, obwohl die Formel korrekt ist. In diesem Fall ist das Programm mit den gleichen Parametern noch einmal zu starten.

5.4 Computer Aided Design (CAD)

5.4.1 Einführung in CAD

Bestimmt standen auch Sie schon einmal vor dem Problem, irgendeine technische Zeichnung anzufertigen. Sie wollten nun Ihren Computer mit seiner herrlichen Grafik für diese Aufgabe einsetzen. Doch schon bald haben Sie dann bestimmt gemerkt, daß dem Commodore 128 irgendwie bestimmte Voraussetzungen fehlen, um diese Aufgabe zu lösen.

An dieser Stelle möchten wir nun an Sie appellieren, sich einmal mit dem "Computer Aided Design" (computerunterstütztes Konstruieren) zu beschäftigen.

Die Hauptaufgabe eines sogenannten CAD-Sytems besteht darin, ein bestimmtes technisches Teil mit Hilfe des Computers auf dem Bildschirm zu konstruieren und später dann auf einem Drucker oder Plotter ausgeben zu lassen. Dabei arbeitet der Computer natürlich um ein Vielfaches schneller, als es ein Mensch jemals schaffen könnte.
In unserem Fall ist das eigentliche Problem die Kommunikation, der Dialog mit dem Rechner. Wir müssen dem Rechner natürlich auf irgendeine Weise mitteilen, an welche Stelle wir welches Teil positioniert haben wollen. Da bestehen für uns leider nicht so viele Möglichkeiten.

Die erste Möglichkeit wäre der Lightpen, der jedoch die Nachteile hat, daß er 1. nicht sehr verbreitet ist und 2. zu ungenau arbeitet.

Zweite Möglichkeit wäre ein Joystick, mit dessen Hilfe wir schon genauer, jedoch auf die Dauer nicht sehr rentabel arbeiten könnten, weil ein solcher Joystick viel zu langsam ist.
Die dritte Möglichkeit wäre ein sogenanntes Grafiktablett, das für zweidimensionale Zeichnungen bestimmt am allerbesten geeignet ist. Kommen wir jedoch zu den dreidimensionalen Gebil-

den, so fehlt uns dann die Möglichkeit, die Raumkoordinate einzuzeichnen.

Bliebe uns als letzte Möglichkeit nur noch die Tastatur, die zwar den Nachteil hat, daß es mitunter eine ganze Zeit braucht, bis man eine bestimmte geometrische Grundfigur positioniert hat, jedoch hat sie den Vorteil, daß jedermann über sie verfügt, und man mit Ihr auch Raumkoordinaten und andere Parameter, wie z. B. die Höhe oder den Radius eines Gebildes, angeben kann.

5.4.2 Einteilung in verschiedene Ebenen

Das nächste, eigentlich nicht grafisch orientierte Problem ist die Einteilung in verschiedene Ebenen. Dazu muß zunächst einmal der Begriff "Ebene" geklärt werden. In bezug auf CAD ist eine Ebene nichts anderes als ein Teil der Gesamtfigur, die wir erstellen wollen. Sie können sich den Begriff "Ebene" am besten vorstellen, wenn Sie ihn mit einer Klarsichtfolie vergleichen, auf der nur ein einziges Stück der Gesamtfigur zu sehen ist. Legt man nun mehrere solcher Figuren, natürlich in einer sinnvollen Anordnung, übereinander, so ergibt sich aus vielen kleinen Einzelteilen eine einzige große Figur.

Große CAD-System haben nun den Vorteil, über sehr viel mehr Speicherplatz zu verfügen, als wir ihn zur Verfügung haben. Sie können einen einzigen Teil, ähnlich wie bei uns die Shapes, zwischenspeichern und brauchen bei Bedarf dieses Shape, oder auch Macro genannt, nur in die Grafik hinein zu kopieren.

Wir haben, wie schon gesagt, wegen akutem Speicherplatzmangel diese Möglichkeit nicht und müssen uns daher etwas anderes einfallen lassen.

Eine Möglichkeit wäre, bei Bedarf jeweils die gesamte Grafik neu zeichnen zu lassen. Damit wäre unser CAD-System natürlich relativ langsam, aber wir hätten auf der anderen Seite eine beliebige Menge Platz für unsere Ebenen.

Wenn wir nun einen "Grafikablauf", also eine Zeichnung anfertigen, so machen wir im Prinzip nicht anderes, als eine Folge von untergeordneten Zeichnungen zu bestimmen. Auch hierzu wieder ein Vergleich, es ist dies das normale BASIC. Das BASIC macht auch nichts anderes, als aufgrund mehrerer Teilinformationen das eigentliche große Programm auszuführen.
Wir müssen natürlich jetzt auch die Möglichkeit haben, Teile (Klarsichtfolien; BASIC-Zeilen) aus dem Gesamt-Zeichenablauf herauszunehmen.

Nun würde sich am besten ein Array als Programmspeicher für unser CAD-System eignen, weil dieser vom BASIC her am einfachsten zu bearbeiten ist.

In diesem Array ist auch das Einfügen oder Löschen von Zeichenschritten nicht mehr sonderlich schwierig. Wir haben für Sie nun ein kleines CAD-System erarbeitet, dessen Funktionsweise wir Ihnen nun nach und nach vorstellen möchten.

Zunächst wäre da das Erstellen eines Gesamt-Menues, mit dem wir später alle Unterfunktionen aufrufen können:

5.4.3 Das Menue

```
100 rem ***********************************
110 rem ***                             ***
120 rem ***        c.a.d. 128           ***
130 rem ***        ----------           ***
140 rem ***                             ***
150 rem ***********************************
160 :
170 dim cm(60,10),p(8) : rem befehlsarrays vorbereiten
180 :
190 rem *** anzahl der parameter festlegen ***
200 :
210 p( 1)=2 : p( 2)=4 : p( 3)=8
```

```
220 p( 4)=6 : p( 5)=3 : p( 6)=4
230 p( 7)=6 : p( 8)=2
240 :
250 xmax=320 : ymax=200 : rem maximale aufloesung
260 :
270 color 0,1           : rem hintergrundfarbe bestimmen
280 color 1,6           : rem zeichen farbe bestimmen
290 color 4,1           : rem rahmenfarbe bestimmen
300 :
310 graphic 1,1         : rem grafik loeschen
320 :
330 rem **********************************
340 rem *** erstellen der zeichnung ****
350 rem **********************************
360 :
370 fast
380 graphic 0
390 print chr$(19)+chr$(19)+chr$(147)+chr$(14)
400 print chr$(27)+"m"+chr$(27)+"e" : rem scrollen verhindern, cursor
steht
410 char ,15,0,"M e n u e"+chr$(13)
420 for x=1 to 40:print "-";:next
430 print:print:print:print "zwei-dimensionale Operationen"
440 print "-----------------------------":print
450 print "Punkt............1    Linie............2"
460 print "Kreis............3    Recheck..........4"
470 print "Rastereinteilung.5    Felder loeschen..6"
480 print "Felder kopieren..7    Flaechen fuellen.8"
490 :
500 for x=1 to 40:print chr$(183);:next:print
510 :
520 slow:zs=1
530 :
540 data "Editieren",1
550 data "Grafik"   ,13
560 data "Diskette" ,21
570 data "Beenden"  ,31
580 :
590 for x=1 to 4          : rem menuepunkte einlesen
600 :read mn$(x),pm(x)    : rem menuepunkt & position
```

```
610 :char ,pm(x),23,mn$(x) : rem auf bildschirm ausgeben
620 next x
630 :
640 char ,1,23,chr$(18)+mn$(1)+chr$(146)
650 :
660 rem ***********************************
670 rem *** menuepunkt auswaehlen ******
680 rem ***********************************
690 :
710 restore:z=1
720 :
730 getkey a$
740 if a$=chr$( 29) then z=z+1:if z>4 then z=1
750 if a$=chr$(157) then z=z-1:if z<1 then z=4
760 if a$=chr$( 13) then 800
770 char ,pm(za),23,chr$(146)+mn$(za)
780 char ,pm( z),23,chr$( 18)+mn$( z)
790 za=z : goto 730
800 print chr$(146);
810 on z goto 810,810,810,810
820 print "S"
830 end
```

Wenn Sie dieses Programm ablaufen lassen, so werden Sie sehen, daß dieses Programm zunächst nur eine Auswahl an CAD-Operationen zur Verfügung stellt. Weiterhin haben Sie die Möglichkeit, mit den Cursortasten einen bestimmten Punkt dieses CAD-Systems auszuwählen und diesen dann durch Bestätigung der "RETURN"-Taste auszuführen.

Sollten Sie zu diesem Zeitpunkt schon einmal die "RETURN"-Taste betätigen, so wird sich der Computer noch "aufhängen", d. h. in einer Warteschleife bleiben, da ja noch keinerlei Menuepunkte vorhanden sind. Dies geschieht in Zeile 810, wo später einmal die Zeilennummern zu den weiteren Menuepunkten stehen müssen.

Das erste Untermenue, das wir nun benötigen, wäre ein Eingabeprogramm. Es müßte grundlegende Dinge können, mit denen

ein sinnvolles Editieren unseres späteren Zeichenablaufes möglich ist.

Dieses Programm könnte nun, passend zu dem ersten Teil des CAD-Sytems wie folgt aussehen:

5.4.4 Das Dialogprogramm

```
840 :
850 rem **********************************
860 rem *** editieren ******************
870 rem **********************************
880 :
890 gosub 1550 : rem statuszeilen loeschen
900 char ,0,23,"1) input  2) delete 3) insert  4) list"+chr$(13)
910 input "Ihre Wahl (5)=ende ";mn
920 if mn<1 or mn>5 then 900 : rem bereichsueberschreitug
930 gosub 1550 : rem statuszeilen loeschen
940 on mn goto 990,1240,1060,1630,540
950 :
960 rem *** zeichenschritt eingeben ***
970 :
980 gosub 1550 : rem statuszeilen loeschen
990 char ,0,23:print "Zeichenschrittnr.:  ";z;"¥¥¥¥¥";:input z
1000 gosub  1380 : rem zeichenart & parameter eingeben
1010 char ,0,23,"Weitere Zeichenschritte eingeben [j/n] ?    "
1020 char ,0,24,"                              ": rem 30 * space
1030 getkey a$:if a$="j" then z=z+1:goto 980
1040 z=z+1:goto 850 : rem zum editier menue
1050 :
1060 rem *** zeichenschritt einfuegen ***
1070 :
1080 gosub 1550 : rem statuszeilen loeschen
1090 char ,0,23,"Vor welchem zeichenschritt ?"+chr$(13)
1100 input z2:gosub 1560 : rem statuszeilen loeschen
1110 if z2>59 then char ,0,23,"Kein Platz vorhanden    ":getkey a$:goto
1090
```

```
1120 fast:for x=z2+1 to 59
1130 :for x2=1 to 10
1140 ::cm(x+1,x2)=cm(x,x2)
1150 next x2,x:za=zs:zs=z2:slow
1160 char ,0,23:print chr$(18)+"Zeichenschrittnr.:";zs;chr$(146)
1170 gosub 1380    : rem zeichenart & parameter eingeben
1180 zs=za
1190 char ,0,23,"Weitere Zeichenschritte einfuegen [j/n] ?"
1200 char ,0,24,"                    ": rem 20 * space
1210 getkey a$:if a$="j" then 1090
1220 goto 850
1230 :
1240 rem *** zeichenschritt loeschen ***
1250 :
1260 gosub 1550 : rem statuszeilen loeschen
1270 char ,0,23,"Welchen Zeichenschritt ?"+chr$(13)
1280 input z2:fast
1290 for x=60 to z2+1 step -1
1300 :for x2=1 to 10
1310 ::cm(x,x2)=cm(x-1,x2)
1320 next x2,x:slow
1330 char ,0,23,"Weitere Zeichenschritte loeschen [j/n] ?"
1340 char ,0,24,"                    ": rem 20 * space
1350 getkey a$:if a$="j" then 1270
1360 goto 850
1370 :
1380 rem **********************************
1390 rem *** schrittart eingeben ********
1400 rem **********************************
1410 :
1420 sleep 1 : rem pause
1430 char ,0,23,"Bitte Zeichenart eingeben"+chr$(13)
1440 input "Zeichenart ";za:cm(z,1)=za
1450 gosub 1550 : rem statuszeilen loeschen
1460 char ,0,23:print "Diese zeichenart benoetigt";p(za);"Parameter"
1470 sleep 1
1480 gosub 1550 : rem statuszeilen loeschen
1490 for x=2 to p(za)+1
1500 :char ,0,23,"Parameternummer :"+str$(x-1)+"       "+chr$(13)
1510 :input cm(z,x)
```

```
1520 next x
1530 return
1540 :
1550 rem **********************************
1560 rem *** zeilen 23 und 24 loeschen **
1570 rem **********************************
1580 :
1590 char ,0,23,"                                        " : rem 40
space
1600 char ,0,24,"                                        " : rem 40
space
1610 return
1620 :
1630 rem **********************************
1640 rem *** programm listen ************
1650 rem **********************************
1660 :
1670 graphic 5:scnclr:print chr$(14)
1680 print "ôauf Drucker oder Bildschirm ? [d/b]"
1690 getkey a$:fast
1700 if a$="d" then open1,4,1:else open1,3
1710 for x=1 to 60
1720 :x$=str$(x)
1730 :print#1,chr$(12) : rem seitenwechsel ausfuehren
1740 :print#1,right$(x$,len(x$)-1)+"       ";:rem 7 * space
1750 :for x2=1 to 9
1760 ::print#1,cm(x,x2);",";
1770 :next x2
1780 :print#1,cm(x,x2)
1790 next x
1800 print:print"Bitte Taste druecken"
1810 getkey a$
1820 close 1
1830 slow:graphic 0
1840 goto 850
1850 :
```

Sie sollten natürlich, bevor Sie sich an diesen Programm heranwagen, die erste Zeilennummer in der Zeile 810 in die Zeile "850" umändern.

Mit diesem zweitem Programmteil haben Sie nun die Möglichkeit, Ihren CAD-Ablauf zu berarbeiten. Dieses Editierprogramm strotzt natürlich nicht gerade vor Eingabefreundlichkeit und Schnelligkeit, aber es zu perfektionieren sollten Sie sich zum Ziel setzen!

Die grundlegenden Dinge jedoch, wie Editieren, Schritt einfügen, Schritt löschen, und Ablauf Listen haben wir in diesen Programmteil integriert.
Beim Listen auf dem Bildschirm mußten wir leider der Einfachkeit halber auf den 80-Zeichenbildschirm ausweichen. Wie schon gesagt, soll dieses Programm ja mehr oder weniger nur eine Anregung Ihrer eigenen Phatasie darstellen.
Doch nun weiter im Text. Es fehlen jetzt noch weitere grundlegende Dinge, die das sinnvolle Arbeiten mit unserem CAD-System ermöglichen.
Natürlich müssen auch Zeichnungen abgespeichert werden können. Die einfachste Lösung dieses Problems wäre, die Grafik als Ganzes abzuspeichern. Leider können wir sie dann nicht mehr richtig editieren, so daß diese Methode als sinnvolle Lösung wegfällt. Aber eine andere Lösung bietet sich an. Da wir mit String-Arrays arbeiten, sollte es kein großes Problem darstellen, diese auf Diskette zu speichern, oder von einer solchen zu laden. Diese Lösung bieten wir Ihnen als weiteres Teilprogramm nun im Folgenden an:

5.4.5 Die I/O-Operationen

```
1860 rem ***********************************
1870 rem *** diskettenoperationen *******
1880 rem ***********************************
1890 :
1900 gosub 1590 : rem statuszeilen loeschen
```

```
1910 char ,0,23,"1) Programm laden  2) Programm speichern"+chr$(13)
1920 input "Ihre Wahl (3=Ende)";a:gosub1590 : rem statuszeilen loeschen
1930 on a goto 2170,1990,530
1940 :
1950 rem ***********************************
1960 rem *** programm speichern *********
1970 rem ***********************************
1980 :
1990 gosub 1590 : rem statuszeilen loeschen
2000 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2010 input dn$
2020 open1,8,1,dn$
2030 for x=1 to 60
2040 :for z=1 to 10
2050 ::print#1,cm(x,z)
2060 next z,x
2070 close 1
2080 gosub 1590
2090 char ,0,23,ds$
2100 char ,0,24,"Bitte taste druecken"
2110 getkey a$:goto 1900
2120 :
2130 rem ***********************************
2140 rem *** programm laden *************
2150 rem ***********************************
2160 :
2170 gosub 1590 : rem statuszeilen loeschen
2180 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2190 input dn$
2200 open1,8,0,dn$
2210 for x=1 to 60
2220 :for z=1 to 10
2230 ::input#1,cm(x,z)
2240 next z,x
2250 close 1
2260 gosub 1590
2270 char ,0,23,ds$
2280 char ,0,24,"Bitte taste druecken"
2290 getkey a$:goto 1900
2300 :
```

Wieder müssen Sie, bevor Sie sich dieses Programmteils bemächtigen, die entsprechende Zeilennummer in der Zeile 810 eintragen. Strikt nach Reihenfolge vorgegangen, der aufmerksame Leser wird dies schon erkannt haben, muß diesmal die dritte Zeilennummer in der Zeile 810 geändert werden.

Eine sinnvolle Erweiterung, die jedoch an dieser Stelle den Rahmen des Buches sprengen würde, wären zusätzlich noch Ausgabeoperationen für Matrixdrucker oder Plotter. Einige Grafikroutinen haben wir ja im Kapitel über die Ein- und Ausgabe von Grafiken, vielleicht haben Sie selber Lust, eine solche zu integrieren?

Nachdem nun alle "nicht-grafikorientierten" Probleme aus dem Weg geräumt wurden, lassen Sie uns nun den grafischen Problemen zuwenden.

Sie haben bestimmt schon im Eingabeprogramm erkannt, daß in unserem Befehlsarray zunächst die Nummer des Befehls gespeichert wurde, und dann die entsprechenden Parameter folgten. Wir können nun unsere Gesamtfigur zeichnen, indem wir das gesamte Array durchgehen, und aufgrund der Nummern der Grafikoperationen die einzelnen Routinen anspringen. Dort können dann in den entsprechenden Routinen die einzelnen Befehle übergeben werden.

Hier ist nun der letzte Teil des CAD-Systems:

5.4.6 Die Grafikroutinen

```
2310 rem ***********************************
2320 rem *** grafikoperationen **********
2330 rem ***********************************
2340 :
2350 gosub 1590 : rem statuszeilen loeschen
```

```
2360 char ,0,23,"Grafik  1) clear  2) scale  3) draw"+chr$(13)
2370 input "Ihre Wahl (4=Ende)";a
2380 on a goto 2450,2500,2660,2390
2390 gosub 1550:goto 540
2400 :
2410 rem ***********************************
2420 rem *** grafik loeschen ************
2430 rem ***********************************
2440 :
2450 gosub 1590 : rem statuszeilen loeschen
2460 char ,0,23,"Grafik loeschen ?"+chr$(13)
2470 input a$:if a$="ja" then gl=1:else gl=0
2480 goto 2350
2490 :
2500 rem ***********************************
2510 rem *** scalierung *****************
2520 rem ***********************************
2530 :
2540 gosub1590
2550 char ,0,23,"X-Aufloesung angeben"+chr$(13)
2560 input xmax
2570 gosub 1590
2580 char ,0,23,"Y-Ausloesung angeben"+chr$(13)
2590 input ymax
2600 goto 2350
2610 :
2620 rem ***********************************
2630 rem *** programm ablaufen lassen ***
2640 rem ***********************************
2650 :
2660 graphic 1,gl
2670 scale 1,xmax,ymax
2680 for x=1 to 60
2690 :s=cm(x,1):if s=0 then 2710
2700 :on s gosub 2850,2920,2990,3060,3130,3240,3310,3380
2710 next x
2720 :
2730 getkey a$
2740 graphic 0
2750 goto 2350
```

```
2760 :
2770 rem *** dummy ***
2780 :
2790 return
2800 :
2810 rem ********************************
2820 rem *** punkt zeichnen *************
2830 rem ********************************
2840 :
2850 draw 1,cm(x,2),cm(x,3)
2860 return
2870 :
2880 rem ********************************
2890 rem *** linie zeichnen *************
2900 rem ********************************
2910 :
2920 draw 1,cm(x,2),cm(x,3) to cm(x,4),cm(x,5)
2930 return
2940 :
2950 rem ********************************
2960 rem *** kreis zeichnen *************
2970 rem ********************************
2980 :
2990 circle 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7),cm(x,8)
3000 return
3010 :
3020 rem ********************************
3030 rem ***  rechteck zeichnen *********
3040 rem ********************************
3050 :
3060 box 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7)
3070 return
3080 :
3090 rem ********************************
3100 rem *** raster einteilung **********
3110 rem ********************************
3120 :
3130 z=0
3140 for xk=1 to xmax step cm(x,2)
3150 for yk=1 to ymax step cm(x,3)
```

```
3160 draw 1,xk,yk
3170 next yk,xk
3180 return
3190 :
3200 rem **********************************
3210 rem *** felder loeschen ************
3220 rem **********************************
3230 :
3240 box 0,cm(x,2),cm(x,3),cm(x,4),cm(x,5),0,1
3250 return
3260 :
3270 rem **********************************
3280 rem *** felder kopieren ************
3290 rem **********************************
3300 :
3310 sshape a$,cm(x,2),cm(x,3),cm(x,4),cm(x,5)
3320 gshape a$,cm(x,6),cm(x,7)
3330 return
3340 :
3350 rem **********************************
3360 rem *** flaechen ausfuellen ********
3370 rem **********************************
3380 :
3390 paint ,cm(x,2),cm(x,3)
3400 return
```

Wie Sie in diesem Listing selbst sehen, haben wir uns nur der Grafikbefehle bedient, die das Commodore-BASIC von vornherein bietet. Selbstverständlich können Sie dieses CAD-System ganz nach Belieben erweitern. Ihrer Phantasie sind hier keine Grenzen gesetzt. Wir wollen Sie jedoch an dieser Stelle auf ein Buch aufmerksam machen, das Ihnen als interessiertem CAD-Anwender und Programmierer bestimmt gefallen wird. Es ist das "CAD-Buch zum Commodore 64", das zwar den Commodore 64 mit der Erweiterung "Simons-BASIC" voraussetzt, Sie dürften aber inzwischen keine Schwierigkeiten mehr haben, diese Befehle auf den Commodore 128 zu übertragen.

Zu guter letzt wollen wir Ihnen das CAD-Programm noch in seiner gesamten Länge auflisten, da durch die Änderungen, die beim Zusammenfügen der einzelnen Teile notwendig sind, Flüchtigkeitsfehler sehr schnell gemacht sind. Auch hier möchten wir Sie gerne auf die Diskette zum Buch verweisen, die Ihnen gerade bei solchen Mammutlistings bestimmt sehr viel Zeit und Ärger erspart.

5.4.7 Das gesamte CAD-System

```
100 rem ***********************************
110 rem ***                             ***
120 rem ***         c.a.d. 128          ***
130 rem ***         ----------          ***
140 rem ***                             ***
150 rem ***********************************
160 :
170 dim cm(60,10),p(8) : rem befehlsarrays vorbereiten
180 :
190 rem *** anzahl der parameter festlegen ***
200 :
210 p( 1)=2 : p( 2)=4 : p( 3)=8
220 p( 4)=6 : p( 5)=3 : p( 6)=4
230 p( 7)=6 : p( 8)=2
240 :
250 xmax=320 : ymax=200 : rem maximale aufloesung
260 :
270 color 0,1            : rem hintergrundfarbe bestimmen
280 color 1,6            : rem zeichen farbe bestimmen
290 color 4,1            : rem rahmenfarbe bestimmen
300 :
310 graphic 1,1          : rem grafik loeschen
320 :
330 rem ***********************************
340 rem *** erstellen der zeichnung ****
350 rem ***********************************
360 :
```

```
370 fast
380 graphic 0
390 print chr$(19)+chr$(19)+chr$(147)+chr$(14)
400 print chr$(27)+"m"+chr$(27)+"e" : rem scrollen verhindern, cursor
steht
410 char ,15,0,"M e n u e"+chr$(13)
420 for x=1 to 40:print "-";:next
430 print:print:print:print "zwei-dimensionale Operationen"
440 print "-----------------------------":print
450 print "Punkt............1    Linie............2"
460 print "Kreis............3    Rechteck.........4"
470 print "Rastereinteilung.5    Felder loeschen..6"
480 print "Felder kopieren..7    Flaechen fuellen.8"
490 :
500 for x=1 to 40:print chr$(183);:next:print
510 :
520 slow:zs=1
530 :
540 data "Editieren",1
550 data "Grafik"   ,13
560 data "Diskette" ,21
570 data "Beenden"  ,31
580 :
590 for x=1 to 4             : rem menuepunkte einlesen
600 :read mn$(x),pm(x)       : rem menuepunkt & position
610 :char ,pm(x),23,mn$(x) : rem auf bildschirm ausgeben
620 next x
630 :
640 char ,1,23,chr$(18)+mn$(1)+chr$(146)
650 :
660 rem ***********************************
670 rem *** menuepunkt auswaehlen ******
680 rem ***********************************
690 :
710 restore:z=1
720 :
730 getkey a$
740 if a$=chr$( 29) then z=z+1:if z>4 then z=1
750 if a$=chr$(157) then z=z-1:if z<1 then z=4
760 if a$=chr$( 13) then 800
```

```
770 char ,pm(za),23,chr$(146)+mn$(za)
780 char ,pm( z),23,chr$( 18)+mn$( z)
790 za=z : goto 730
800 print chr$(146);
810 on z goto 850,2350,1860,820
820 print "S"
830 end
840 :
850 rem ***********************************
860 rem *** editieren ******************
870 rem ***********************************
880 :
890 gosub 1550 : rem statuszeilen loeschen
900 char ,0,23,"1) input  2) delete 3) insert  4) list"+chr$(13)
910 input "Ihre Wahl (5)=ende ";mn
920 if mn<1 or mn>5 then 900 : rem bereichsueberschreitug
930 gosub 1550 : rem statuszeilen loeschen
940 on mn goto 990,1240,1060,1630,540
950 :
960 rem *** zeichenschritt eingeben ***
970 :
980 gosub 1550 : rem statuszeilen loeschen
990 char ,0,23:print "Zeichenschrittnr.:  ";z;"¥¥¥¥¥";:input z
1000 gosub  1380 : rem zeichenart & parameter eingeben
1010 char ,0,23,"Weitere Zeichenschritte eingeben [j/n] ?    "
1020 char ,0,24,"                              ": rem 30 * space
1030 getkey a$:if a$="j" then z=z+1:goto 980
1040 z=z+1:goto 850 : rem zum editier menue
1050 :
1060 rem *** zeichenschritt einfuegen ***
1070 :
1080 gosub 1550 : rem statuszeilen loeschen
1090 char ,0,23,"Vor welchem zeichenschritt ?"+chr$(13)
1100 input z2:gosub 1560 : rem statuszeilen loeschen
1110 if z2>59 then char ,0,23,"Kein Platz vorhanden    ":getkey a$:goto
1090
1120 fast:for x=z2+1 to 59
1130 :for x2=1 to 10
1140 ::cm(x+1,x2)=cm(x,x2)
1150 next x2,x:za=zs:zs=z2:slow
```

```
1160 char ,0,23:print chr$(18)+"Zeichenschrittnr.:";zs;chr$(146)
1170 gosub 1380   : rem zeichenart & parameter eingeben
1180 zs=za
1190 char ,0,23,"Weitere Zeichenschritte einfuegen [j/n] ?"
1200 char ,0,24,"                    ": rem 20 * space
1210 getkey a$:if a$="j" then 1090
1220 goto 850
1230 :
1240 rem *** zeichenschritt loeschen ***
1250 :
1260 gosub 1550 : rem statuszeilen loeschen
1270 char ,0,23,"Welchen Zeichenschritt ?"+chr$(13)
1280 input z2:fast
1290 for x=60 to z2+1 step -1
1300 :for x2=1 to 10
1310 ::cm(x,x2)=cm(x-1,x2)
1320 next x2,x:slow
1330 char ,0,23,"Weitere Zeichenschritte loeschen [j/n] ?"
1340 char ,0,24,"                    ": rem 20 * space
1350 getkey a$:if a$="j" then 1270
1360 goto 850
1370 :
1380 rem **********************************
1390 rem *** schrittart eingeben ********
1400 rem **********************************
1410 :
1420 sleep 1 : rem pause
1430 char ,0,23,"Bitte Zeichenart eingeben"+chr$(13)
1440 input "Zeichenart ";za:cm(z,1)=za
1450 gosub 1550 : rem statuszeilen loeschen
1460 char ,0,23:print "Diese zeichenart benoetigt";p(za);"Parameter"
1470 sleep 1
1480 gosub 1550 : rem statuszeilen loeschen
1490 for x=2 to p(za)+1
1500 :char ,0,23,"Parameternummer :"+str$(x-1)+"       "+chr$(13)
1510 :input cm(z,x)
1520 next x
1530 return
1540 :
1550 rem **********************************
```

```
1560 rem *** zeilen 23 und 24 loeschen **
1570 rem *********************************
1580 :
1590 char ,0,23,"                                        " : rem 40
space
1600 char ,0,24,"                                        " : rem 40
space
1610 return
1620 :
1630 rem *********************************
1640 rem *** programm listen ************
1650 rem *********************************
1660 :
1670 graphic 5:scnclr:print chr$(14)
1680 print "auf Drucker oder Bildschirm ? d/b"
1690 getkey a$:fast
1700 if a$="d" then open1,4,1:else open1,3
1710 for x=1 to 60
1720 :x$=str$(x)
1730 :print#1,chr$(12) : rem seitenwechsel ausfuehren
1740 :print#1,right$(x$,len(x$)-1)+"       ";:rem 7 * space
1750 :for x2=1 to 9
1760 ::print#1,cm(x,x2);",";
1770 :next x2
1780 :print#1,cm(x,x2)
1790 next x
1800 print:print"Bitte Taste druecken"
1810 getkey a$
1820 close 1
1830 slow:graphic 0
1840 goto 850
1850 :
1860 rem *********************************
1870 rem *** diskettenoperationen *******
1880 rem *********************************
1890 :
1900 gosub 1590 : rem statuszeilen loeschen
1910 char ,0,23,"1) Programm laden  2) Programm speichern"+chr$(13)
1920 input "Ihre Wahl (3=Ende)";a:gosub1590 : rem statuszeilen loeschen
1930 on a goto 2170,1990,530
```

```
1940 :
1950 rem **********************************
1960 rem *** programm speichern *********
1970 rem **********************************
1980 :
1990 gosub 1590 : rem statuszeilen loeschen
2000 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2010 input dn$
2020 open1,8,1,dn$
2030 for x=1 to 60
2040 :for z=1 to 10
2050 ::print#1,cm(x,z)
2060 next z,x
2070 close 1
2080 gosub 1590
2090 char ,0,23,ds$
2100 char ,0,24,"Bitte taste druecken"
2110 getkey a$:goto 1900
2120 :
2130 rem **********************************
2140 rem *** programm laden *************
2150 rem **********************************
2160 :
2170 gosub 1590 : rem statuszeilen loeschen
2180 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2190 input dn$
2200 open1,8,0,dn$
2210 for x=1 to 60
2220 :for z=1 to 10
2230 ::input#1,cm(x,z)
2240 next z,x
2250 close 1
2260 gosub 1590
2270 char ,0,23,ds$
2280 char ,0,24,"Bitte taste druecken"
2290 getkey a$:goto 1900
2300 :
2310 rem **********************************
2320 rem *** grafikoperationen **********
2330 rem **********************************
```

```
2340 :
2350 gosub 1590 : rem statuszeilen loeschen
2360 char ,0,23,"Grafik  1) clear  2) scale  3) draw"+chr$(13)
2370 input "Ihre Wahl (4=Ende)";a
2380 on a goto 2450,2500,2660,2390
2390 gosub 1550:goto 540
2400 :
2410 rem *********************************
2420 rem *** grafik loeschen ************
2430 rem *********************************
2440 :
2450 gosub 1590 : rem statuszeilen loeschen
2460 char ,0,23,"Grafik loeschen ?"+chr$(13)
2470 input a$:if a$="ja" then gl=1:else gl=0
2480 goto 2350
2490 :
2500 rem *********************************
2510 rem *** scalierung *****************
2520 rem *********************************
2530 :
2540 gosub1590
2550 char ,0,23,"X-Aufloesung angeben"+chr$(13)
2560 input xmax
2570 gosub 1590
2580 char ,0,23,"Y-Ausloesung angeben"+chr$(13)
2590 input ymax
2600 goto 2350
2610 :
2620 rem *********************************
2630 rem *** programm ablaufen lassen ***
2640 rem *********************************
2650 :
2660 graphic 1,gl
2670 scale 1,xmax,ymax
2680 for x=1 to 60
2690 :s=cm(x,1):if s=0 then 2710
2700 :on s gosub 2850,2920,2990,3060,3130,3240,3310,3380
2710 next x
2720 :
2730 getkey a$
```

```
2740 graphic 0
2750 goto 2350
2760 :
2770 rem *** dummy ***
2780 :
2790 return
2800 :
2810 rem ********************************
2820 rem *** punkt zeichnen *************
2830 rem ********************************
2840 :
2850 draw 1,cm(x,2),cm(x,3)
2860 return
2870 :
2880 rem ********************************
2890 rem *** linie zeichnen *************
2900 rem ********************************
2910 :
2920 draw 1,cm(x,2),cm(x,3) to cm(x,4),cm(x,5)
2930 return
2940 :
2950 rem ********************************
2960 rem *** kreis zeichnen *************
2970 rem ********************************
2980 :
2990 circle 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7),cm(x,8)
3000 return
3010 :
3020 rem ********************************
3030 rem ***  rechteck zeichnen *********
3040 rem ********************************
3050 :
3060 box 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7)
3070 return
3080 :
3090 rem ********************************
3100 rem *** raster einteilung **********
3110 rem ********************************
3120 :
3130 z=0
```

```
3140 for xk=1 to xmax step cm(x,2)
3150 for yk=1 to ymax step cm(x,3)
3160 draw 1,xk,yk
3170 next yk,xk
3180 return
3190 :
3200 rem ***********************************
3210 rem *** felder loeschen ************
3220 rem ***********************************
3230 :
3240 box 0,cm(x,2),cm(x,3),cm(x,4),cm(x,5),0,1
3250 return
3260 :
3270 rem ***********************************
3280 rem *** felder kopieren ************
3290 rem ***********************************
3300 :
3310 sshape a$,cm(x,2),cm(x,3),cm(x,4),cm(x,5)
3320 gshape a$,cm(x,6),cm(x,7)
3330 return
3340 :
3350 rem ***********************************
3360 rem *** flaechen ausfuellen ********
3370 rem ***********************************
3380 :
3390 paint ,cm(x,2),cm(x,3)
3400 return
```

5.5 Ausgabe von Grafiken

5.5.1 Hardcopy RX80/FX80 40 Zeichen

Hat man anspruchsvolle und komplizierte Grafiken erstellt, besteht des öfteren das Bedürfnis, diese schwarz auf weiß zu haben. Für diesen Zweck, also die Übertragung des Grafikschirms an einen Drucker, existieren die sogenannten Hardcopyroutinen. Von diesen Routinen wollen wir Ihnen zwei vorstel-

len. Eine für die 40-Zeichengrafik und eine für die 80-Zeichengrafik. Hier zuerst die 40-Zeichenhardcopy:

Assemblerlisting:

```
profi-ass 64 v2.0

120:    1300                .opt p1,oo
130:    1300                *=   $1300
140:                    ;
150:    1300 48             pha
160:    1301 a9 00          lda  #<8192
160:    1303 a0 20          ldy  #>8192
170:    1305 85 fa          sta  $fa
170:    1307 84 fb          sty  $fb       ; basis adresse setzen
180:                    ;
190:    1309 68             pla
190:    130a 48             pha
200:    130b a0 01          ldy  #1        ; sekundaeradresse
210:    130d 20 ba ff       jsr  $ffba     ; setpara
220:    1310 a9 00          lda  #0
230:    1312 20 bd ff       jsr  $ffbd     ; setnam
240:    1315 20 c0 ff       jsr  $ffc0     ; open
250:    1318 68             pla
260:    1319 48             pha
270:    131a aa             tax
280:    131b 20 c9 ff       jsr  $ffc9     ; chkout
290:                    ;
300:    131e a9 1b          lda  #27       ; escape "1" senden,
310:    1320 20 d2 ff       jsr  $ffd2     ; dadurch zeilenabstand
320:    1323 a9 31          lda  #"1"      ; setzen
330:    1325 20 d2 ff       jsr  $ffd2
340:                    ;
350:    1328 20 45 13       jsr  hardcopy  ; hardcopy drucken
360:                    ;
370:    132b a9 1b          lda  #27       ; escape "2" senden,
380:    132d 20 d2 ff       jsr  $ffd2     ; dadurch zeilenabstand
390:    1330 a9 32          lda  #"2"      ; normalisieren
400:    1332 20 d2 ff       jsr  $ffd2
```

```
410:    1335 20 cc ff            jsr  $ffcc    ; clrch
420:    1338 68                  pla
430:    1339 4c c3 ff            jmp  $ffc3    ; close
440:                    ;
450:    133c 00         bytes    .byt %00000000
460:    133d 00                  .byt %00000000
470:    133e 00                  .byt %00000000
480:    133f 00                  .byt %00000000
490:    1340 00                  .byt %00000000
500:    1341 00                  .byt %00000000
510:    1342 00                  .byt %00000000
520:    1343 00                  .byt %00000000
530:                    ;
540:    1344 00         spalte   .byt 0        ; spaltenzaehler
550:                    ;
560:    1345 a2 18      hardcopy ldx  #24      ; zeilenzaehler setzen
570:    1347 a9 0d      lab1     lda  #13
570:    1349 20 d2 ff            jsr  $ffd2    ; zeilenvorschub
580:    134c a0 04               ldy  #4
590:    134e b9 75 13   loop     lda  drcktab,y  ; steuerzeichen
600:    1351 20 d2 ff            jsr  $ffd2    ; ausgeben
610:    1354 88                  dey
610:    1355 10 f7               bpl  loop
620:                    ;
630:    1357 a9 27               lda  #39      ; spaltenzaehler
640:    1359 8d 44 13            sta  spalte   ; setzen
650:    135c a0 00      lab2     ldy  #0       ; y-register loeschen
660:    135e 20 7a 13            jsr  sendacht  ; 8ter block ausgeben
670:    1361 a5 fa               lda  $fa      ; $fa/$fb = $fa/$fb + 8
680:    1363 18                  clc
690:    1364 69 08               adc  #8
700:    1366 90 02               bcc  ok1
710:    1368 e6 fb               inc  $fb
720:    136a 85 fa      ok1      sta  $fa
730:    136c ce 44 13            dec  spalte   ; spalte=spalte-1
740:    136f 10 eb               bpl  lab2
750:    1371 ca                  dex
760:    1372 10 d3               bpl  lab1
770:    1374 60                  rts
780:                    ;
```

```
790:    1375 02 80 4c drcktab  .byt 2,128,76,27,9
800:                  ;
810:                  ;*** drehen und setzen eines 8ter blocks ***
820:                  ;
830:    137a 8a       sendacht txa
830:    137b 48                pha
830:    137c 98                tya
830:    137d 48                pha
840:    137e a9 80             lda  #$80
850:    1380 8d b7 13          sta  maske
860:    1383 a2 07             ldx  #7
870:    1385 a0 00    hrdloop1 ldy  #0
880:    1387 b1 fa    hrdloop2 lda  ($fa),y
890:    1389 2d b7 13          and  maske
900:    138c 38                sec
910:    138d d0 01             bne  ok2
920:    138f 18                clc
930:    1390 3e 3c 13 ok2      rol  bytes,x
940:    1393 c8                iny
950:    1394 c0 08             cpy  #8
960:    1396 d0 ef             bne  hrdloop2
970:    1398 4e b7 13          lsr  maske
980:    139b ca                dex
980:    139c 10 e7             bpl  hrdloop1
990:                  ;
1000:   139e a2 07             ldx  #7       ; bytes ausgeben
1010:   13a0 bd 3c 13 outlab   lda  bytes,x
1020:   13a3 20 d2 ff          jsr  $ffd2
1020:   13a6 20 d2 ff          jsr  $ffd2
1020:   13a9 a9 00             lda  #0
1020:   13ab 9d 3c 13          sta  bytes,x
1030:   13ae ca                dex
1040:   13af 10 ef             bpl  outlab
1050:   13b1 68                pla
1050:   13b2 a8                tay
1050:   13b3 68                pla
1050:   13b4 aa                tax
1050:   13b5 60                rts
1060:                 ;
```

```
1070:  13b6 00       temp     .byt 0
1080:  13b7 00       maske    .byt 0
]1300-13b8
```

BASIC-Lader:

```
100 rem ***********************************************
110 rem ***                                         ***
120 rem ***  hardcopy des 40 zeichenbildschirms     ***
130 rem ***  ---------------------------------- ***
140 rem ***                                         ***
150 rem ***     (basic-loader)                      ***
160 rem ***                                         ***
170 rem ***********************************************
180 :
190 for x=4864 to 5067
200 :read a
210 :poke x,a
220 :p=p+a
230 next x
240 :
250 if p<>23588 then print " fehler in datas":end
260 :
270 print " datas sind ok. aufruf mit 'sys 4864,lf,geraet"
280 end
290 :
300 data 072,169,000,160,032,133,250,132,251
310 data 251,104,072,160,001,032,186,255,169
320 data 169,000,032,189,255,032,192,255,104
330 data 104,072,170,032,201,255,169,027,032
340 data 032,210,255,169,049,032,210,255,032
350 data 032,069,019,169,027,032,210,255,169
360 data 169,050,032,210,255,032,204,255,104
370 data 104,076,195,255,000,000,000,000,000
380 data 000,000,000,000,000,162,024,169,013
390 data 013,032,210,255,160,004,185,117,019
400 data 019,032,210,255,136,016,247,169,039
410 data 039,141,068,019,160,000,032,122,019
420 data 019,165,250,024,105,008,144,002,230
```

```
430 data 230,251,133,250,206,068,019,016,235
440 data 235,202,016,211,096,002,128,076,027
450 data 027,009,138,072,152,072,169,128,141
460 data 141,183,019,162,007,160,000,177,250
470 data 250,045,183,019,056,208,001,024,062
480 data 062,060,019,200,192,008,208,239,078
490 data 078,183,019,202,016,231,162,007,189
500 data 189,060,019,032,210,255,032,210,255
510 data 255,169,000,157,060,019,202,016,239
520 data 239,104,168,104,170,096
```

5.5.2 Hardcopy RX80/FX80 80 Zeichen

Assemblerlisting:

```
profi-ass 64 v2.0

110:   1400                      .opt p,oo
120:   1400                      *=   $1400
130:                    ;
140:                    ;*********************************
150:                    ;***                           ***
160:                    ;***  hardcopy vdc-grafik      ***
170:                    ;***  -------------------      ***
180:                    ;***                           ***
190:                    ;***  auf allen epson rx/fx    ***
200:                    ;***  compt. nadeldruckern     ***
210:                    ;***                           ***
220:                    ;*********************************
230:                    ;
240:   1400 4c 32 14             jmp  hardcopy  ; zur hardnot-routine
250:                    ;
260:   1403 8d 00 d6 setreg      sta  $d600    ; register setzen
270:   1406 2c 00 d6 loop1       bit  $d600
270:   1409 10 fb                bpl  loop1
280:   140b 8e 01 d6             stx  $d601
290:   140e 60                   rts
```

```
300:                    ;
310:   140f 8d 00 d6 getreg    sta  $d600    ; register auslesen
320:   1412 2c 00 d6 loop2     bit  $d600
320:   1415 10 fb              bpl  loop2
330:   1417 ad 01 d6           lda  $d601
340:   141a 60                 rts
350:                    ;
360:   141b 48       setadr    pha           ; adresse setzen
360:   141c a9 12              lda  #18
360:   141e 20 03 14           jsr  setreg
370:   1421 68                 pla
370:   1422 aa                 tax
370:   1423 a9 13              lda  #19
370:   1425 4c 03 14           jmp  setreg
380:                    ;
390:   1428 00       lincnt    .byt 0        ; zeilenzaehler
400:    1429 03 20 4c tab1      .byt 3,32,76,27,32,32,32,32,32  ; steuer-
zeichen in umgek. reihenfolge
410:                    ;
420:                    ;
430:   1432 48       hardcopy pha
430:   1433 a9 00              lda  #0
430:   1435 85 fa              sta  $fa
440:   1437 85 fb              sta  $fb
470:   1439 a0 01              ldy  #1
470:   143b 68                 pla
470:   143c 48                 pha
480:   143d 20 ba ff           jsr  $ffba
490:   1440 a9 00              lda  #0
490:   1442 20 bd ff           jsr  $ffbd
500:   1445 20 c0 ff           jsr  $ffc0
510:   1448 68                 pla
510:   1449 48                 pha
510:   144a aa                 tax
510:   144b 20 c9 ff           jsr  $ffc9    ; file oeffnen
520:                    ;
530:   144e a9 1b              lda  #27
530:   1450 20 d2 ff           jsr  $ffd2    ; zeilenabstand auf 7/72 zoll
540:   1453 a9 31              lda  #49
540:   1455 20 d2 ff           jsr  $ffd2
```

```
550:  1458 a9 4f              lda  #79
550:  145a 8d 28 14           sta  lincnt  ; line-counter setzen
560:                    ;
570:  145d a9 0d     lab1     lda  #13
570:  145f 20 d2 ff           jsr  $ffd2   ; 1. schleife
580:  1462 a2 08              ldx  #8
590:  1464 bd 29 14 loop3     lda  tab1,x
590:  1467 20 d2 ff           jsr  $ffd2
590:  146a ca                 dex
600:  146b 10 f7              bpl  loop3
600:  146d ad 28 14           lda  lincnt
600:  1470 85 fa              sta  $fa
610:  1472 a9 00              lda  #0
610:  1474 85 fb              sta  $fb
620:                    ;
630:  1476 a0 c8              ldy  #200
640:  1478 a5 fa     lab2     lda  $fa
640:  147a a6 fb              ldx  $fb
640:  147c 20 1b 14           jsr  setadr
650:  147f a9 1f              lda  #31
650:  1481 20 0f 14           jsr  getreg
660:  1484 8d c8 14           sta  mem
660:  1487 8a                 txa
660:  1488 48                 pha
670:  1489 a2 00              ldx  #0
670:  148b ad c8 14           lda  mem
680:  148e 2a        schloop  rol
680:  148f 6e c8 14           ror  mem
690:  1492 e8                 inx
690:  1493 e0 08              cpx  #8
690:  1495 d0 f7              bne  schloop
700:  1497 68                 pla
700:  1498 aa                 tax
710:  1499 ad c8 14           lda  mem
720:  149c 20 d2 ff           jsr  $ffd2
720:  149f 20 d2 ff           jsr  $ffd2
720:  14a2 20 d2 ff           jsr  $ffd2   ; 4 bytes ausgeben, dadurch
730:  14a5 20 d2 ff           jsr  $ffd2   ; grafik entzerren
740:  14a8 a5 fa              lda  $fa
740:  14aa 18                 clc
```

```
740:    14ab 69 50              adc  #80
740:    14ad 85 fa              sta  $fa
750:    14af 90 02              bcc  ok1
750:    14b1 e6 fb              inc  $fb
760:    14b3 88         ok1     dey
760:    14b4 d0 c2              bne  lab2
770:    14b6 ce 28 14           dec  lincnt
770:    14b9 10 a2              bpl  lab1     ; 80 zeilen ausgeben
780:    14bb 20 c9 14           jsr  reset    ; zeilen abstand setzen
790:    14be 68                 pla
790:    14bf 48                 pha
790:    14c0 aa                 tax
790:    14c1 20 cc ff           jsr  $ffcc
790:    14c4 68                 pla
800:    14c5 4c c3 ff           jmp  $ffc3
810:                    ;
820:    14c8 00         mem     .byt 0
830:                    ;
840:    14c9 a9 1b      reset   lda  #27
840:    14cb 20 d2 ff           jsr  $ffd2    ; escape
850:    14ce a9 32              lda  #"2"     ; "2" ausgeben, damit zeilen-
verschub
860:    14d0 4c d2 ff           jmp  $ffd2    ; normalisieren
]1400-14d3
```

BASIC-Lader:

```
100 rem **********************************************
110 rem ***                                        ***
120 rem ***  hardcopy des 80 zeichenbildschirms    ***
130 rem ***  ------------------------------------- ***
140 rem ***                                        ***
150 rem ***     (basic-loader)                     ***
160 rem ***                                        ***
170 rem **********************************************
180 :
190 for x=5120 to 5331
200 :read a
210 :poke x,a
```

```
220 :p=p+a
230 next x
240 :
250 if p<>25058 then print " fehler in datas":end
260 :
270 print " datas sind ok. aufruf mit 'sys 5120,lf,geraet"
280 end
290 :
300 data 076,050,020,141,000,214,044,000,214
310 data 214,016,251,142,001,214,096,141,000
320 data 000,214,044,000,214,016,251,173,001
330 data 001,214,096,072,169,018,032,003,020
340 data 020,104,170,169,019,076,003,020,070
350 data 070,003,032,076,027,032,032,032,032
360 data 032,032,072,169,000,133,250,133,251
370 data 251,160,001,104,072,032,186,255,169
380 data 169,000,032,189,255,032,192,255,104
390 data 104,072,170,032,201,255,169,027,032
400 data 032,210,255,169,049,032,210,255,169
410 data 169,079,141,040,020,169,013,032,210
420 data 210,255,162,008,189,041,020,032,210
430 data 210,255,202,016,247,173,040,020,133
440 data 133,250,169,000,133,251,160,200,165
450 data 165,250,166,251,032,027,020,169,031
460 data 031,032,015,020,141,200,020,138,072
470 data 072,162,000,173,200,020,042,110,200
480 data 200,020,232,224,008,208,247,104,170
490 data 170,173,200,020,032,210,255,032,210
500 data 210,255,032,210,255,032,210,255,165
510 data 165,250,024,105,080,133,250,144,002
520 data 002,230,251,136,208,194,206,040,020
530 data 020,016,162,032,201,020,104,072,170
540 data 170,032,204,255,104,076,195,255,255
550 data 255,169,027,032,210,255,169,050,076
560 data 076,210,255,000,255,000,255,000,255
```

5.6 Grafikprogrammierung in 6502-Assembler

5.6.1 Supergrafik 128

Bestimmt waren Sie enttäuscht, als Sie im Kapitel über den VDC die sehr gute Grafik kennenlernten, jedoch um so mehr die Langsamkeit der Beispielprogramme erkannten. Doch das soll nun anders werden. Im folgenden legen wir Ihnen ein Programm in die Hände, das Ihnen bestimmt gefallen wird. Da es vollkommen in Assembler geschrieben ist, sind die einzelnen Grafikroutinen auch dementsprechend schnell.
Leider läuft das Programm nur im 64er-Modus, wo, wie Sie ja vielleicht wissen, die VDC-Grafik auch ansprechbar ist. Dies mußte aus Gründen der Sparsamkeit geschehen, damit noch Raum für weitere Anwendungen blieb.

Sie selber sehen, daß dieses Programm mit seinen zehn Seiten Assemblerlisting nicht gerade das kürzeste ist, jedoch lohnt sich das Abtippen des weiter unten stehenden BASIC-Laders ganz bestimmt. Das Assemblerlisting haben wir beigefügt, um den Profis unter Ihnen einen "Anstoß" zum eigenen Programmieren zu geben.

Sollten Sie keine Zeit oder Lust haben, dieses Programm abzutippen, so kann ich Ihnen an dieser Stelle die "Diskette zum großen Grafikbuch" empfehlen. Jetzt jedoch zuerst einmal das Assembler-Listing, das übrigens wieder mit dem Profi-Ass 64 assembliert wurde.

5.6.2 Assembler-Listing

Achtung, dieses Programm läuft nur im 64er Modus!

```
110:  c000                    .opt p,oo
120:  c000                    .sym 4        ; vierspaltige symboltabelle
130:  c000                    .tit "extended graphic 128"  ; titel
140:  c000                    *=   $c000    ; assemblierung ab $c000
150:               ;
160:  aefd         chkcom     =    $aefd
170:  b79e         getbyt     =    $b79e
180:  b7eb         getadr     =    $b7eb
190:  d600         vdc        =    $d600
200:  ffd2         print      =    $ffd2
210:  cf00         msk        =    $cf00
220:  cf01         dif0       =    $cf01
230:  cf02         dif1       =    $cf02
240:  cf03         dif2       =    $cf03
250:  cf04         dif3       =    $cf04
260:  cf05         dif4       =    $cf05
270:  cf06         dif5       =    $cf06
280:  cf07         zwn        =    $cf07
290:  cf08         za         =    $cf08
300:  cf09         a          =    $cf09
310:  cf0a         b          =    $cf0a
320:  cf0b         yk         =    $cf0b
330:  cf0c         temp       =    $cf0c
340:               ;
350:               ;
360:               ;************************************
370:               ;***    extended graphic 128     ***
380:               ;***      written 1985 by        ***
390:               ;***         axel plenge         ***
400:               ;***             and             ***
410:               ;***      klaus loeffelmann      ***
420:               ;*** (!! nur im 64er modus !!)   ***
430:               ;************************************
440:               ;
```

```
450:                    ;*** einschaltmeldung ausgeben ***
460:                    ;
470:   c000 a2 00               ldx  #0        ; x-register loeschen
480:   c002 bd 1c c0 loop1      lda  asctab1,x  ; buchstaben laden
490:   c005 20 d2 ff            jsr  print     ; und ausgeben
500:   c008 f0 04               beq  endl1     ; 0-byte, dann fertig
510:   c00a e8                  inx            ; x-register erhoehen
520:   c00b 4c 02 c0            jmp  loop1     ; zum schleifenanfang
530:                    ;
540:   c00e a9 00    endl1      lda  #0        ; hintergrund schwarz
550:   c010 8d 20 d0            sta  $d020
560:   c013 8d 21 d0            sta  $d021
570:   c016 20 d9 c0            jsr  gron      ; grafik einschalten
580:   c019 4c e8 c0            jmp  grclr
590:                    ;
600:   c01c 93       asctab1    .byt 147       ; bildschirm loeschen
610:   c01d 0d                  .byt 13        ; punktfarbe gruen
620:   c01e 2d 2d 2d             .asc "-------------------------------------
---"
620:   c045 0d                  .byt 13
630:   c046 3e 3e 3e               .asc ">>>           extended graphic 128
<<<"
630:   c06d 0d                  .byt 13
640:   c06e 3e 3e 3e              .asc ">>>  autoren :  loeffelmann/plenge
<<<"
640:   c095 0d                  .byt 13
650:   c096 2d 2d 2d             .asc "-------------------------------------
---"
650:   c0bd 0d                  .byt 13
660:   c0be 0d 0d 00            .byt 13,13,0
670:                    ;
680:                    ;*** register setzen ***
690:                    ;
700:   c0c1 8d 00 d6 regset     sta  vdc
710:   c0c4 2c 00 d6 loop2      bit  vdc
710:   c0c7 10 fb               bpl  loop2
720:   c0c9 8e 01 d6            stx  vdc+1
730:   c0cc 60                  rts
740:                    ;
750:                    ;*** register auslesen ***
```

```
760:                     ;
770:  c0cd 8d 00 d6 regred    sta  vdc
780:  c0d0 2c 00 d6 loop3     bit  vdc
780:  c0d3 10 fb              bpl  loop3
790:  c0d5 ad 01 d6           lda  vdc+1
800:  c0d8 60                 rts
810:                     ;
820:                     ;*** grafik einschalten ***
830:                     ;
840:  c0d9 a9 19    gron      lda  #25       ; register auswaehlen
840:  c0db 48                 pha            ; reg. und zwischspchn.
850:  c0dc 20 cd c0           jsr  regred    ; register 25 auslesen
860:  c0df 09 80              ora  #%10000000  ; bit 7 setzen
870:  c0e1 29 bf              and  #%10111111  ; bit 6 loeschen
880:  c0e3 aa                 tax            ; ins x-register bringen
890:  c0e4 68                 pla            ; registernummer vom stack
900:  c0e5 4c c1 c0           jmp  regset
910:                     ;
920:                     ;*** grafik loeschen ***
930:                     ;
940:  c0e8 a9 00    grclr     lda  #0
940:  c0ea a2 00              ldx  #0        ; up-date adresse
950:  c0ec 20 23 c1           jsr  setadr    ; auf null setzen
960:  c0ef a9 1f              lda  #31
960:  c0f1 a2 00              ldx  #0        ; data register loeschen
970:  c0f3 20 c1 c0           jsr  regset    ; register setzen
980:                     ;
990:  c0f6 a0 3e              ldy  #62       ; 16128 bytes loeschen
1000: c0f8 a9 1e              lda  #30       ; no.des word-count reg.
1010: c0fa a2 00              ldx  #0        ; wert f.word-count reg.
1020:                    ;
1030: c0fc 20 c1 c0 loop4     jsr  regset    ; register setzen
1040: c0ff 88                 dey
1050: c100 10 fa              bpl  loop4     ; das ganze 63 mal
1060: c102 60                 rts
1070:                    ;
1080:                    ;*** basicbefehl "color" ***
1090:                    ;
1100: c103 20 fd ae color     jsr  chkcom
1110: c106 20 9e b7           jsr  getbyt
```

```
1120:  c109 8a                txa
1120:  c10a 0a                asl
1120:  c10b 0a                asl
1120:  c10c 0a                asl
1120:  c10d 0a                asl
1120:  c10e 8d 22 c1          sta  colstr
1130:  c111 20 fd ae          jsr  chkcom
1140:  c114 20 9e b7          jsr  getbyt
1140:  c117 8a                txa
1150:  c118 18                clc
1150:  c119 6d 22 c1          adc  colstr  ; =hf*16+pf
1160:  c11c aa                tax
1160:  c11d a9 1a             lda  #26     ; in register 26
1170:  c11f 4c c1 c0          jmp  regset  ; abspeichern
1180:  c122 00       colstr   .byt 0
1190:                ;
1200:                ;*** adresse setzen ***
1210:                ;
1220:  c123 48       setadr   pha          ; low-byte auf stack
1230:  c124 a9 12             lda  #18     ; high-byte der adresse
1240:  c126 20 c1 c0          jsr  regset  ; setzen
1250:  c129 68                pla          ; low-byte vom stack
1260:  c12a aa                tax          ; und ins x-register
1270:  c12b a9 13             lda  #19     ; register 19
1280:  c12d 4c c1 c0          jmp  regset  ; setzen
1290:                ;
1300:  c130 00 00    xk       .wor 0
1310:  c132 00       xkl      .byt 0
1320:  c133 00       xkh      .byt 0
1330:  c134 00       flag     .byt 0
1340:                ;
1350:                ;*** adresse eines punktes berechnen ***
1360:                ;
1370:  c135 a9 00    calcadr  lda  #0      ; high-byte adresse loeschen
1380:  c137 8d 0a cf          sta  b
1390:  c13a ad 0b cf          lda  yk      ; y-koord. als lb der adr.
1400:                ;
1410:  c13d 0a                asl          ; *2
1420:  c13e 2e 0a cf          rol  b
1430:  c141 0a                asl          ; *2 (=*4)
```

```
1440:  c142 2e 0a cf          rol  b
1450:  c145 18                clc
1460:  c146 6d 0b cf          adc  yk     ; +1 (=*5)
1470:  c149 90 03             bcc  ok1    ; ueberlauf beruecksichtigen
1480:  c14b ee 0a cf          inc  b
1490:  c14e 0a         ok1    asl
1490:  c14f 2e 0a cf          rol  b
1500:  c152 0a                asl
1500:  c153 2e 0a cf          rol  b
1510:  c156 0a                asl
1510:  c157 2e 0a cf          rol  b
1520:  c15a 0a                asl
1520:  c15b 2e 0a cf          rol  b
1530:  c15e 8d 09 cf          sta  a      ; *16 (=*80)
1540:                  ;
1550:  c161 a6 15             ldx  $15
1550:  c163 8e 33 c1          stx  xkh    ; x-koordinate nach
1560:  c166 a5 14             lda  $14
1560:  c168 48                pha
1560:  c169 8d 32 c1          sta  xkl    ; xkl/xkh bringen
1570:  c16c 8a                txa
1580:                  ;
1590:  c16d 4a                lsr
1590:  c16e 66 14             ror  $14
1600:  c170 4a                lsr
1600:  c171 66 14             ror  $14
1610:  c173 4a                lsr
1610:  c174 66 14             ror  $14    ; und durch  teilen
1620:  c176 48                pha         ; high-byte aus stack
1630:  c177 a5 14             lda  $14    ; x-koordinate zur
1640:  c179 18                clc
1650:  c17a 6d 09 cf          adc  a      ; adresse addieren
1660:  c17d 90 03             bcc  ok2
1670:  c17f ee 0a cf          inc  b
1680:  c182 8d 09 cf ok2      sta  a
1690:  c185 68                pla         ; high-byte vom stack
1700:  c186 18                clc
1710:  c187 6d 0a cf          adc  b      ; high-bytes addieren
1720:  c18a 8d 0a cf          sta  b
1730:                  ;
```

```
1740:                   ;* maske errechnen *
1750:                   ;
1760:  c18d 68                  pla           ; x-low vom stack holen
1770:  c18e 29 07               and  #7
1780:  c190 aa                  tax           ; ins x-register bringen
1790:  c191 bd 98 c1            lda  msktab,x
1800:  c194 8d 00 cf            sta  msk      ; = 2^(7-(x and 7))
1810:  c197 60                  rts
1820:                   ;
1830:  c198 80 40 20 msktab     .byt 128,64,32,16,8,4,2,1
1840:                   ;
1850:                   ;*** punkt plotten ***
1860:                   ;
1870:  c1a0 a0 00    dot1       ldy  #0
1870:  c1a2 08                  php
1870:  c1a3 8a                  txa
1870:  c1a4 48                  pha
1870:  c1a5 ad 09 cf            lda  a
1880:  c1a8 ae 0a cf            ldx  b        ; punkt adresse setzen
1890:  c1ab 20 23 c1            jsr  setadr
1900:  c1ae a9 1f               lda  #31
1900:  c1b0 20 cd c0            jsr  regred   ; bitmuster auslesen
1910:  c1b3 ac 34 c1            ldy  flag     ; sdot/cdot/idot - flag
1920:  c1b6 f0 10               beq  cdot     ; flag=0, dann cdot
1930:  c1b8 c0 01               cpy  #1
1940:  c1ba f0 06               beq  sdot     ; flag=1, dann sdot
1950:  c1bc 4d 00 cf            eor  msk      ; flag=2, dann idot
1960:  c1bf 4c d3 c1            jmp  store
1970:  c1c2 0d 00 cf sdot       ora  msk      ; sdot
1980:  c1c5 4c d3 c1            jmp  store
1990:  c1c8 8d 0c cf cdot       sta  temp     ; cdot
2000:  c1cb ad 00 cf            lda  msk
2000:  c1ce 49 ff               eor  #$ff
2000:  c1d0 2d 0c cf            and  temp
2010:                   ;
2020:  c1d3 48       store      pha
2030:  c1d4 ad 09 cf            lda  a
2030:  c1d7 ae 0a cf            ldx  b        ; adresse setzen
2040:  c1da 20 23 c1            jsr  setadr
2050:  c1dd 68                  pla
```

```
2050:  c1de aa                tax          ; wert ins x-register
2060:  c1df a9 1f             lda  #31     ; punkt
2070:  c1e1 20 c1 c0          jsr  regset  ; abspeichern
2080:  c1e4 ac 07 cf          ldy  zwn
2080:  c1e7 68                pla
2080:  c1e8 aa                tax
2080:  c1e9 28                plp
2080:  c1ea 60                rts
2090:                  ;
2100:                  ;*** basicbefehl "dot" ***
2110:                  ;
2120:  c1eb 20 fd ae dot2     jsr  chkcom  ; komma holen
2130:  c1ee 20 9e b7          jsr  getbyt  ; modus
2140:  c1f1 8e 34 c1          stx  flag
2150:  c1f4 20 fd ae          jsr  chkcom
2160:  c1f7 20 eb b7          jsr  getadr  ; 16 und 8 bitwert holen
2170:  c1fa 8e 0b cf          stx  yk
2180:  c1fd 20 35 c1          jsr  calcadr  ; adresse errechnen
2190:  c200 4c a0 c1          jmp  dot1
2200:                  ;
2210:                  ;*** vektorroutinen ***
2220:                  ;
2230:  c203 ad 09 cf unten    lda  a       ; adresse + 80
2240:  c206 69 50             adc  #80
2240:  c208 8d 09 cf          sta  a
2250:  c20b 90 03             bcc  ok3
2260:  c20d ee 0a cf          inc  b
2270:  c210 60       ok3      rts
2280:                  ;
2290:  c211 30 f0    uo       bmi  unten
2300:                  ;
2310:  c213 ad 09 cf oben     lda  a       ; adresse - 80
2320:  c216 38                sec
2320:  c217 e9 50             sbc  #80
2320:  c219 8d 09 cf          sta  a
2330:  c21c b0 03             bcs  ok4
2340:  c21e ce 0a cf          dec  b
2350:  c221 60       ok4      rts
2360:                  ;
2370:  c222 4e 00 cf rechts   lsr  msk     ; adresse +1
```

```
2380:  c225 90 0d              bcc  ok5
2390:  c227 c8                 iny
2400:  c228 6e 00 cf           ror  msk
2410:  c22b 18                 clc
2410:  c22c ee 09 cf           inc  a
2420:  c22f d0 03              bne  ok5
2430:  c231 ee 0a cf           inc  b
2440:  c234 60        ok5      rts
2450:                 ;
2460:  c235 10 eb     rl       bpl  rechts
2470:                 ;
2480:  c237 0e 00 cf  links    asl  msk       ; adresse -1
2490:  c23a 90 11              bcc  ok6
2500:  c23c 88                 dey
2500:  c23d 2e 00 cf           rol  msk
2510:  c240 ce 09 cf           dec  a
2510:  c243 ad 09 cf           lda  a
2510:  c246 c9 ff              cmp  #255
2520:  c248 d0 03              bne  ok6
2530:  c24a ce 0a cf           dec  b
2540:  c24d 18        ok6      clc
2540:  c24e 60                 rts
2550:                 ;
2560:                 ;*** linie zeichnen ***
2570:                 ;
2580:  c24f 48        hline    pha            ; x1-low in xkl
2590:  c250 ad 33 c1           lda  xkh       ; x2-high in xkh
2600:  c253 4a                 lsr            ; y1 in yk
2610:  c254 ad 32 c1           lda  xkl       ; x2-low im akku
2620:  c257 6a                 ror            ; x2-high im x-reg
2620:  c258 4a                 lsr            ; y2 im y-reg
2620:  c259 4a                 lsr
2620:  c25a 8d 07 cf           sta  zwn       ; x1/8 zwischspei.
2630:  c25d 68                 pla
2630:  c25e 48                 pha
2630:  c25f 38                 sec
2640:  c260 ed 32 c1           sbc  xkl
2650:  c263 48                 pha
2660:  c264 8a                 txa
2670:  c265 ed 33 c1           sbc  xkh
```

```
2680:  c268 8d 04 cf        sta  dif3    ; x2-x1 negativ
2690:  c26b b0 0b           bcs  l3      ; ja
2700:  c26d 68              pla
2710:  c26e 49 ff           eor  #$ff
2720:  c270 69 01           adc  #1
2730:  c272 48              pha
2740:  c273 a9 00           lda  #0
2750:  c275 ed 04 cf        sbc  dif3    ; vorzeichenwechsel
2760:  c278 8d 02 cf l3     sta  dif1
2770:  c27b 8d 06 cf        sta  dif5    ; (x2-x1) high
2780:  c27e 68              pla
2790:  c27f 8d 01 cf        sta  dif0
2800:  c282 8d 05 cf        sta  dif4    ; (x2-x1) low
2810:  c285 68              pla
2820:  c286 8d 32 c1        sta  xkl
2830:  c289 8e 33 c1        stx  xkh
2840:  c28c 98              tya
2850:  c28d 18              clc
2860:  c28e ed 0b cf        sbc  yk      ; y2-y1
2870:  c291 90 04           bcc  l4
2880:  c293 49 ff           eor  #$ff
2890:  c295 69 fe           adc  #$fe
2900:  c297 8d 03 cf l4     sta  dif2
2910:  c29a 8c 0b cf        sty  yk
2920:  c29d 6e 04 cf        ror  dif3
2930:  c2a0 38              sec
2940:  c2a1 ed 01 cf        sbc  dif0
2950:  c2a4 aa              tax
2960:  c2a5 a9 ff           lda  #$ff
2970:  c2a7 ed 02 cf        sbc  dif1
2980:  c2aa 8d 08 cf        sta  za
2990:  c2ad ac 07 cf        ldy  zwn
3000:  c2b0 b0 05           bcs  l5
3010:  c2b2 0a        l1    asl
3020:  c2b3 20 35 c2        jsr  rl
3030:  c2b6 38              sec
3040:  c2b7 ad 05 cf l5     lda  dif4
3050:  c2ba 6d 03 cf        adc  dif2
3060:  c2bd 8d 05 cf        sta  dif4
3070:  c2c0 ad 06 cf        lda  dif5
```

```
3080:  c2c3 e9 00              sbc  #0
3090:  c2c5 8d 06 cf l2        sta  dif5
3100:  c2c8 8c 07 cf           sty  zwn
3110:  c2cb 20 a0 c1           jsr  dot1
3120:  c2ce e8                 inx
3130:  c2cf d0 06              bne  l6
3140:  c2d1 ee 08 cf           inc  za
3150:  c2d4 d0 01              bne  l6
3160:  c2d6 60                 rts
3170:  c2d7 ad 04 cf l6        lda  dif3
3180:  c2da b0 d6              bcs  l1
3190:  c2dc 20 11 c2           jsr  uo
3200:  c2df 18                 clc
3210:  c2e0 ad 05 cf           lda  dif4
3220:  c2e3 6d 01 cf           adc  dif0
3230:  c2e6 8d 05 cf           sta  dif4
3240:  c2e9 ad 06 cf           lda  dif5
3250:  c2ec 6d 02 cf           adc  dif1
3260:  c2ef 50 d4              bvc  l2
3270:                  ;
3280:                  ;
3290:                  ;*** basic-befehl "linie" ***
3300:                  ;
3310:  c2f1 20 eb c1 linie     jsr  dot2
3320:  c2f4 20 fd ae           jsr  chkcom
3330:  c2f7 20 eb b7           jsr  getadr
3340:  c2fa 8a                 txa
3350:  c2fb a8                 tay
3360:  c2fc a5 14              lda  $14
3370:  c2fe a6 15              ldx  $15
3380:  c300 4c 4f c2           jmp  hline
3390:                  ;
3400:                  ;*** basic-befehl "box" ***
3410:                  ;
3420:  c303 00 00    boxxk1    .wor 0
3430:  c305 00       boxyk1    .byt 0
3440:  c306 00 00    boxxk2    .wor 0
3450:  c308 00       boxyk2    .byt 0
3460:                  ;
3470:  c309 20 eb c1 box       jsr  dot2     ; ersten punkt errechen
```

```
3480:  c30c a5 14                lda  $14
3480:  c30e a4 15                ldy  $15
3490:  c310 8d 03 c3             sta  boxxk1
3490:  c313 8c 04 c3             sty  boxxk1+1
3500:  c316 ad 0b cf             lda  yk
3500:  c319 8d 05 c3             sta  boxyk1
3510:  c31c 20 fd ae             jsr  chkcom
3520:  c31f 20 eb b7             jsr  getadr
3530:                    ;
3540:  c322 a5 14                lda  $14
3540:  c324 a4 15                ldy  $15
3550:  c326 8d 06 c3             sta  boxxk2
3550:  c329 8c 07 c3             sty  boxxk2+1
3560:  c32c 8e 08 c3             stx  boxyk2
3570:                    ;
3580:  c32f ad 03 c3 boxlab      lda  boxxk1
3580:  c332 85 14                sta  $14
3590:  c334 ad 04 c3             lda  boxxk1+1
3590:  c337 85 15                sta  $15
3600:  c339 ad 05 c3             lda  boxyk1
3600:  c33c 8d 0b cf             sta  yk
3610:  c33f 20 35 c1             jsr  calcadr
3620:  c342 ad 03 c3             lda  boxxk1
3620:  c345 8d 32 c1             sta  xkl
3630:  c348 ad 04 c3             lda  boxxk1+1
3640:  c34b 8d 33 c1             sta  xkh
3650:  c34e ad 05 c3             lda  boxyk1
3660:  c351 8d 0b cf             sta  yk
3670:  c354 ad 06 c3             lda  boxxk2
3670:  c357 ae 07 c3             ldx  boxxk2+1
3670:  c35a ac 05 c3             ldy  boxyk1
3680:                    ;
3690:  c35d 20 4f c2             jsr  hline
3700:                    ;
3710:  c360 ee 05 c3             inc  boxyk1
3720:  c363 ad 05 c3             lda  boxyk1
3730:  c366 cd 08 c3             cmp  boxyk2
3740:  c369 d0 c4                bne  boxlab
3750:                    ;
3760:  c36b 60                   rts
```

```
3770:                     ;
3780:                     ;*** getstring ***
3790:                     ;
3800: c36c 20 fd ae getstr    jsr  $aefd
3800: c36f 18                 clc
3810: c370 20 9e ad           jsr  $ad9e
3820: c373 20 8f ad           jsr  $ad8f
3830: c376 4c a6 b6           jmp  $b6a6
3840:                     ;
3850:                     ;**********************************
3860:                     ;*** input/output operationen ****
3870:                     ;**********************************
3880:                     ;
3890:                     ;* konstanten *
3900:                     ;
3910: ffc0                open     =    $ffc0
3920: ffba                setfls   =    $ffba
3930: ffbd                setnam   =    $ffbd
3940: ffc9                chkout   =    $ffc9
3950: ffcc                clrch    =    $ffcc
3960: ffcf                input    =    $ffcf
3970: ffc6                chkin    =    $ffc6
3980: ffc3                close    =    $ffc3
3990:                     ;
4000:                     ;*** "open 1,8,y,"prgnam" formulieren" ***
4010:                     ;
4020: c379 a9 01          open18x  lda  #1      ; logische filenummer im akku
4030: c37b a2 08                   ldx  #8      ; geraeteadresse im x-reg
4040: c37d 20 ba ff                jsr  setfls  ; sekundadr. im y-reg
4050: c380 ad 8d c3                lda  strlan  ; akku stringlaenge
4060: c383 a6 22                   ldx  $22     ; low-byte stringadresse
4070: c385 a4 23                   ldy  $23     ; high-byte stringadresse
4080: c387 20 bd ff                jsr  setnam  ; namen setzen
4090: c38a 4c c0 ff                jmp  open    ; datei oeffnen
4100:                     ;
4110: c38d 00             strlan   .byt 0
4120:                     ;
4130:                     ;*** grafik abspeichern ***
4140:                     ;
4150: c38e 20 6c c3 grvsav   jsr  getstr  ; string holen
```

```
4160:  c391 8d 8d c3          sta  strlan  ; stringlaenge abspeichern
4170:  c394 a0 01             ldy  #1
4170:  c396 20 79 c3          jsr  open18x  ; datei oeffnen
4170:  c399 a2 01             ldx  #1
4170:  c39b 20 c9 ff          jsr  chkout  ; cmd 1
4180:                  ;
4190:  c39e a9 00             lda  #0
4190:  c3a0 a2 00             ldx  #0
4190:  c3a2 20 23 c1          jsr  setadr  ; adresse auf 0 setzen
4200:                  ;
4210:  c3a5 a2 3e             ldx  #62     ; 16128 bytes abspeichern
4220:  c3a7 a0 00             ldy  #0      ; y-reg. loeschen
4230:                  ;
4240:  c3a9 a9 1f      gslab  lda  #31
4250:  c3ab 20 cd c0          jsr  regred  ; grafikbyte holen
4260:  c3ae 20 d2 ff          jsr  print   ; und abspeichern
4270:  c3b1 c8                iny
4280:  c3b2 d0 f5             bne  gslab
4290:  c3b4 ca                dex
4300:  c3b5 10 f2             bpl  gslab
4310:  c3b7 4c e8 c3          jmp  grcls
4320:                  ;
4330:                  ;*** grafik laden ***
4340:                  ;
4350:  c3ba 20 6c c3 grvlad   jsr  getstr  ; string holen
4360:  c3bd 8d 8d c3          sta  strlan  ; stringlaenge abspeichern
4370:  c3c0 a0 00             ldy  #0
4370:  c3c2 20 79 c3          jsr  open18x  ; datei oeffnen
4370:  c3c5 a2 01             ldx  #1
4370:  c3c7 20 c6 ff          jsr  chkin   ; cmd 1
4380:                  ;
4390:  c3ca a9 00             lda  #0
4390:  c3cc a2 00             ldx  #0
4390:  c3ce 20 23 c1          jsr  setadr  ; adresse auf 0 setzen
4400:                  ;
4410:  c3d1 a2 3e             ldx  #62     ; 63*256=16128 bytes laden
4420:  c3d3 a0 00             ldy  #0      ; y-reg. loeschen
4430:                  ;
4440:  c3d5 8a         gllab  txa
4440:  c3d6 48                pha
```

```
4450:  c3d7 20 cf ff           jsr  input   ; ein byte holen
4460:  c3da aa                 tax          ; ins x-register bringen
4470:  c3db a9 1f              lda  #31     ; data register laden
4480:  c3dd 20 c1 c0           jsr  regset  ; register setzen
4490:  c3e0 68                 pla
4490:  c3e1 aa                 tax
4500:  c3e2 c8                 iny
4500:  c3e3 d0 f0              bne  gllab
4510:  c3e5 ca                 dex
4520:  c3e6 10 ed              bpl  gllab
4530:                    ;
4540:  c3e8 20 cc ff grcls     jsr  clrch
4550:  c3eb a9 01              lda  #1
4550:  c3ed 4c c3 ff           jmp  close
```

Symboltabelle:

```
grcls    c3e8  gllab    c3d5  grvlad   c3ba  gslab    c3a9
grvsav   c38e  strlan   c38d  open18x  c379  close    ffc3
chkin    ffc6  input    ffcf  clrch    ffcc  chkout   ffc9
setnam   ffbd  setfls   ffba  open     ffc0  getstr   c36c
boxlab   c32f  box      c309  boxyk2   c308  boxxk2   c306
boxyk1   c305  boxxk1   c303  linie    c2f1  l6       c2d7
l2       c2c5  l5       c2b7  l1       c2b2  l4       c297
l3       c278  hline    c24f  ok6      c24d  links    c237
rl       c235  ok5      c234  rechts   c222  ok4      c221
oben     c213  uo       c211  ok3      c210  unten    c203
dot2     c1eb  store    c1d3  cdot     c1c8  sdot     c1c2
dot1     c1a0  msktab   c198  ok2      c182  ok1      c14e
calcadr  c135  flag     c134  xkh      c133  xkl      c132
xk       c130  setadr   c123  colstr   c122  color    c103
loop4    c0fc  grclr    c0e8  gron     c0d9  loop3    c0d0
regred   c0cd  loop2    c0c4  regset   c0c1  asctab1  c01c
endl1    c00e  loop1    c002  temp     cf0c  yk       cf0b
b        cf0a  a        cf09  za       cf08  zwn      cf07
dif5     cf06  dif4     cf05  dif3     cf04  dif2     cf03
dif1     cf02  dif0     cf01  msk      cf00  print    ffd2
vdc      d600  getadr   b7eb  getbyt   b79e  chkcom   aefd
```

5.6.3 BASIC-Lader

Hier ist nun der versprochene BASIC-Lader, der nicht nur das Abtippen erheblich erleichtert, sondern auch dem Assembler-unerfahrenen die Möglichkeit bietet, sich dieses Programm-Paketes zu bedienen.

Zu diesem Lader ist nun noch folgendes zu sagen: Er selbst arbeitet im 128er Modus, um das Eintippen der Datas (Sie können hier die Funktionstasten zu Hilfe nehmen) zu erleichtern und zu beschleunigen.

Sollten Sie das Programm fehlerfrei abgetippt haben, so erscheint die Aufforderung:
"PROGRAMMNAME?"

Tippen Sie nun den Namen ein, unter dem Ihr Programm auf Diskette gespeichert werden soll. Anschließend können Sie die Befehlsfolge

"GO 64"

eingeben, um in den 64er Modus zu gelangen. Der Rechner stellt Ihnen dann noch einmal eine Sicherheitsabfrage ("ARE YOU SURE?"), die Sie nach Möglichkeit mit "JA" beantworten sollten. Dann dauert es eine kleine Zeit, bis Sie sich im 64er Modus befinden, da der Rechner einen kompletten Neustart des Gerätes durchführt, und ihn auf den 64er Modus vorbereitet.

Sind Sie erst einmal im 64er Modus angekommen, so können Sie Ihr Programm mit

LOAD "PRGNAM",8,1

von Diskette einladen. Anschließend müssen Sie durch den Befehl

NEW

die Programmzeiger, die durch einen Fehler im Betriebssystem auch beim Laden von Assemblerprogrammen gesetzt werden, auf den ursprünglichen Stand gebracht werden.
Haben Sie erst einmal Ihr Programm auf Diskette gespeichert, so können die ersten Schritte natürlich entfallen.
Zugegeben, einfach ist dies alles nicht, aber wir glauben das nichts "schief gehen kann", wenn Sie alles genau nach Anweisung befolgen.

Hier ist nun das BASIC-Listing:

```
1000 rem ***********************************
1010 rem ***                             ***
1020 rem ***    extended grafik 128      ***
1030 rem ***    ------------------       ***
1040 rem ***                             ***
1050 rem *** (basic lader)               ***
1060 rem ***                             ***
1070 rem ***********************************
1080 forx=49152to50161
1090 read a$ : a=dec(a$)
1100 p=p+a
1110 pokex,a
1120 next x
1130 if p<>106367 then print "fehler in datas":end
1140 input "programmname";pn$
1150 bsave (pn$),b0,p49152 to p50162
1160 end
1170 :
1180 rem assembler datas
1190 :
1200 data a2,00,bd,1c,c0,20,d2,ff,f0,04
1210 data e8,4c,02,c0,a9,00,8d,20,d0,8d
1220 data 21,d0,20,d9,c0,4c,e8,c0,93,0d
1230 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1240 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1250 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1260 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,0d
```

```
1270 data 3e,3e,3e,20,20,20,20,20,20,20
1280 data 20,45,58,54,45,4e,44,45,44,20
1290 data 47,52,41,50,48,49,43,20,31,32
1300 data 38,20,20,20,20,20,3c,3c,3c,0d
1310 data 3e,3e,3e,20,20,41,55,54,4f,52
1320 data 45,4e,20,3a,20,20,4c,4f,45,46
1330 data 46,45,4c,4d,41,4e,4e,2f,50,4c
1340 data 45,4e,47,45,20,20,3c,3c,3c,0d
1350 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1360 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1370 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,2d
1380 data 2d,2d,2d,2d,2d,2d,2d,2d,2d,0d
1390 data 0d,0d,00,8d,00,d6,2c,00,d6,10
1400 data fb,8e,01,d6,60,8d,00,d6,2c,00
1410 data d6,10,fb,ad,01,d6,60,a9,19,48
1420 data 20,cd,c0,09,80,29,bf,aa,68,4c
1430 data c1,c0,a9,00,a2,00,20,23,c1,a9
1440 data 1f,a2,00,20,c1,c0,a0,3e,a9,1e
1450 data a2,00,20,c1,c0,88,10,fa,60,20
1460 data fd,ae,20,9e,b7,8a,0a,0a,0a,0a
1470 data 8d,22,c1,20,fd,ae,20,9e,b7,8a
1480 data 18,6d,22,c1,aa,a9,1a,4c,c1,c0
1490 data 00,48,a9,12,20,c1,c0,68,aa,a9
1500 data 13,4c,c1,c0,00,00,00,00,00,a9
1510 data 00,8d,0a,cf,ad,0b,cf,0a,2e,0a
1520 data cf,0a,2e,0a,cf,18,6d,0b,cf,90
1530 data 03,ee,0a,cf,0a,2e,0a,cf,0a,2e
1540 data 0a,cf,0a,2e,0a,cf,0a,2e,0a,cf
1550 data 8d,09,cf,a6,15,8e,33,c1,a5,14
1560 data 48,8d,32,c1,8a,4a,66,14,4a,66
1570 data 14,4a,66,14,48,a5,14,18,6d,09
1580 data cf,90,03,ee,0a,cf,8d,09,cf,68
1590 data 18,6d,0a,cf,8d,0a,cf,68,29,07
1600 data aa,bd,98,c1,8d,00,cf,60,80,40
1610 data 20,10,08,04,02,01,a0,00,08,8a
1620 data 48,ad,09,cf,ae,0a,cf,20,23,c1
1630 data a9,1f,20,cd,c0,ac,34,c1,f0,10
1640 data c0,01,f0,06,4d,00,cf,4c,d3,c1
1650 data 0d,00,cf,4c,d3,c1,8d,0c,cf,ad
1660 data 00,cf,49,ff,2d,0c,cf,48,ad,09
```

```
1670 data cf,ae,0a,cf,20,23,c1,68,aa,a9
1680 data 1f,20,c1,c0,ac,07,cf,68,aa,28
1690 data 60,20,fd,ae,20,9e,b7,8e,34,c1
1700 data 20,fd,ae,20,eb,b7,8e,0b,cf,20
1710 data 35,c1,4c,a0,c1,ad,09,cf,69,50
1720 data 8d,09,cf,90,03,ee,0a,cf,60,30
1730 data f0,ad,09,cf,38,e9,50,8d,09,cf
1740 data b0,03,ce,0a,cf,60,4e,00,cf,90
1750 data 0d,c8,6e,00,cf,18,ee,09,cf,d0
1760 data 03,ee,0a,cf,60,10,eb,0e,00,cf
1770 data 90,11,88,2e,00,cf,ce,09,cf,ad
1780 data 09,cf,c9,ff,d0,03,ce,0a,cf,18
1790 data 60,48,ad,33,c1,4a,ad,32,c1,6a
1800 data 4a,4a,8d,07,cf,68,48,38,ed,32
1810 data c1,48,8a,ed,33,c1,8d,04,cf,b0
1820 data 0b,68,49,ff,69,01,48,a9,00,ed
1830 data 04,cf,8d,02,cf,8d,06,cf,68,8d
1840 data 01,cf,8d,05,cf,68,8d,32,c1,8e
1850 data 33,c1,98,18,ed,0b,cf,90,04,49
1860 data ff,69,fe,8d,03,cf,8c,0b,cf,6e
1870 data 04,cf,38,ed,01,cf,aa,a9,ff,ed
1880 data 02,cf,8d,08,cf,ac,07,cf,b0,05
1890 data 0a,20,35,c2,38,ad,05,cf,6d,03
1900 data cf,8d,05,cf,ad,06,cf,e9,00,8d
1910 data 06,cf,8c,07,cf,20,a0,c1,e8,d0
1920 data 06,ee,08,cf,d0,01,60,ad,04,cf
1930 data b0,d6,20,11,c2,18,ad,05,cf,6d
1940 data 01,cf,8d,05,cf,ad,06,cf,6d,02
1950 data cf,50,d4,20,eb,c1,20,fd,ae,20
1960 data eb,b7,8a,a8,a5,14,a6,15,4c,4f
1970 data c2,00,00,00,00,00,00,20,eb,c1
1980 data a5,14,a4,15,8d,03,c3,8c,04,c3
1990 data ad,0b,cf,8d,05,c3,20,fd,ae,20
2000 data eb,b7,a5,14,a4,15,8d,06,c3,8c
2010 data 07,c3,8e,08,c3,ad,03,c3,85,14
2020 data ad,04,c3,85,15,ad,05,c3,8d,0b
2030 data cf,20,35,c1,ad,03,c3,8d,32,c1
2040 data ad,04,c3,8d,33,c1,ad,05,c3,8d
2050 data 0b,cf,ad,06,c3,ae,07,c3,ac,05
2060 data c3,20,4f,c2,ee,05,c3,ad,05,c3
```

```
2070 data cd,08,c3,d0,c4,60,20,fd,ae,18
2080 data 20,9e,ad,20,8f,ad,4c,a6,b6,a9
2090 data 01,a2,08,20,ba,ff,ad,8d,c3,a6
2100 data 22,a4,23,20,bd,ff,4c,c0,ff,00
2110 data 20,6c,c3,8d,8d,c3,a0,01,20,79
2120 data c3,a2,01,20,c9,ff,a9,00,a2,00
2130 data 20,23,c1,a2,3e,a0,00,a9,1f,20
2140 data cd,c0,20,d2,ff,c8,d0,f5,ca,10
2150 data f2,4c,e8,c3,20,6c,c3,8d,8d,c3
2160 data a0,00,20,79,c3,a2,01,20,c6,ff
2170 data a9,00,a2,00,20,23,c1,a2,3e,a0
2180 data 00,8a,48,20,cf,ff,aa,a9,1f,20
2190 data c1,c0,68,aa,c8,d0,f0,ca,10,ed
2200 data 20,cc,ff,a9,01,4c,c3,ff,60,9f
```

Anhang

Anhang -A- Tabellierung aller Escape-Funktionen

ESC ESC - Der automatische Einfügemodus wird ausgeschaltet. Ebenso der Hochkomma- (Quote) und Revers-Modus.

ESC . - Der Bildschirm wird ab der momentanen Cursorposition bis zur linken unteren Bildschirm bzw. Fensterecke gelöscht.

ESC A - Der automatische Einfügemodus wird eingeschaltet, d.h. werden Zeichen eingegeben, so werden diese in den bestehenden Text eingefügt und dieser nicht überschrieben.

ESC B - Durch diese Tastenkombination wird die momentane Cursorposition als rechte, untere Ecke eines Bildschirmfensters definiert.

ESC C - Der durch ESC A aktivierte automatische Einfügemodus wird wieder ausgeschaltet, also der Überschreibmodus aktiviert.

ESC D - Die aktuelle Bildschirmzeile wird gelöscht und der Bildschirmbereich unter der momentanen Cursorposition um eine Zeile nach oben verschoben.

ESC E - Das Cursorblinken wird ausgeschaltet und auf Daueranzeige umgeschaltet.

ESC F - Das Cursorblinken wird eingeschaltet.

ESC G - Die Tastenkombination CTRL-G wird erlaubt und führt in Folge zu einem Klingeln.

ESC H - Die Tastenkombination CTRL-G wird verboten und hat in Folge keine Bedeutung mehr.

ESC I - Fügt oberhalb der Cursorposition eine neue Zeile ein und rollt das Bildschirmfenster um eine Zeile nach unten.

ESC J - Der Cursor wird an der Anfang der aktuellen Bildschirmzeile gesetzt.

ESC K - Der Cursor wird hinter das letzte Zeichen der aktuellen Zeile gesetzt.
ESC L - Das Bildschirmscrolling wird erlaubt.
ESC M - Das Bildschirmscrolling wird verboten. Bei Überschreitung der untersten Zeile wird ein Cursor-Home ausgeführt.
ESC N - Ist der 80-Zeichenschirm aktiviert, so wird auf Normal-Darstellung umgeschaltet. Diese Tastenkombination besitzt im 40-Zeichen-Modus keine Wirkung.
ESC O - Diese Tastenkombination besitzt die gleiche Funktion wie die weiter oben beschriebene Tastenkombination ESC-ESC.
ESC P - Die momentane Zeile wird vom Zeilenanfang bis zur Cursorposition gelöscht.
ESC Q - Die momentane Zeile wird von der Cursorposition bis zum Zeilenende gelöscht.
ESC R - Ein eventuell angeschlossener 80-Zeichen RGBI-Monitor wird auf INVERS-Darstellung umgeschaltet. Desaktivieren kann man diesen betriebsmodus durch die weiter oben beschreibene Tastenkombination ESC N.
ESC S - Es wird ein Blockcursor dargestellt.
ESC T - Die momentane Cursorposition wird als linke obere Ecke eines Windows definiert.
ESC U - Der Cursor auf dem 80-Zeichenschirm wird als Strich-Cursor dargestellt.
ESC V - Der gesamte Bildschirm oder ein eventuell bestehendes Window wird um eine Zeile nach oben verschoben.
ESC W - Der gesamte Bildschirm oder ein Window wird um eine Zeile nach unten verschoben.
ESC X - Als Ausgabegerät dient der 80-Zeichenschirm, falls der 40-Zeichenschirm momentan aktiviert war und umgekehrt.
ESC Y - Stellt die Tabulatoren auf jeweils acht Spalten ein.
ESC Z - Löscht alle vordefinierten Tabulatorstops.

Anhang -B- Tabellierung aller Control-Funktionen

CTRL -B- & CHR$(2):	Alle Zeichen auf dem 80-Zeichenmonitor werden unterstrichen angezeigt.
CTRL -G- & CHR$(7):	Ein akustisches Klingelzeichen wird ausgelöst.
CTRL -I- & CHR$(9):	Der Cursor wird auf den nächsten Tabulatorstop gesetzt.
CTRL -J- & CHR$(10):	Zeilenvorschub.
CTRL -K- & CHR$(11):	Die Umschaltung Klein/Groß und Groß/Grafik wird verboten.
CTRL -L- & CHR$(12):	Die Umschaltung zwischen den beiden Alternativzeichensätzen wird wieder erlaubt.
CTRL -O- & CHR$(15):	Alle Zeichen auf dem 80-Zeichenmonitor werden blinkend dargestellt.
CTRL -X- & CHR$(24):	Ein Tabulatorstop wird gesetzt oder gelöscht.
CTRL -[- & CHR$(27):	Escape.

Anhang -C- ASCII- und Bildschirmcodetabellen

Chr$ & Poke Kodes

Dez	Hex	POKE- Zeichen Gross	POKE- Zeichen Klein	CHR$- Zeichen Gross	CHR$- Zeichen Klein
ØØØ	$ØØ	@	@	Keine Funktion	
ØØ1	$Ø1	A	a	Keine Funktion	
ØØ2	$Ø2	B	b	Keine Funktion	
ØØ3	$Ø3	C	c	Keine Funktion	
ØØ4	$Ø4	D	d	Keine Funktion	
ØØ5	$Ø5	E	e	Weiss	
ØØ6	$Ø6	F	f	Keine Funktion	
ØØ7	$Ø7	G	g	Keine Funktion	
ØØ8	$Ø8	H	h	Shift Commodore verboten	
ØØ9	$Ø9	I	i	Shift Commodore erlauben	
Ø1Ø	$Øa	J	j	Keine Funktion	
Ø11	$Øb	K	k	Keine Funktion	
Ø12	$Øc	L	l	Keine Funktion	
Ø13	$Ød	M	m	Return	
Ø14	$Øe	N	n	Kleinbuchstaben	
Ø15	$Øf	O	o	Keine Funktion	
Ø16	$1Ø	P	p	Keine Funktion	
Ø17	$11	Q	q	Cursor abwaerts	
Ø18	$12	R	r	Revers anschalten	
Ø19	$13	S	s	Cursor Home	
Ø2Ø	$14	T	t	Delete	
Ø21	$15	U	u	Keine Funktion	
Ø22	$16	V	v	Keine Funktion	
Ø23	$17	W	w	Keine Funktion	
Ø24	$18	X	x	Keine Funktion	
Ø25	$19	Y	y	Keine Funktion	
Ø26	$1a	Z	z	Keine Funktion	
Ø27	$1b	[	[	Escape	
Ø28	$1c	£	£	Rot	
Ø29	$1d	]	]	Cursor rechts	
Ø3Ø	$1e	↑	↑	Gruen	
Ø31	$1f	←	←	Dunkelblau	
Ø32	$2Ø			Space	
Ø33	$21	!	!	!	!
Ø34	$22			"	"
Ø35	$23	#	#	#	#
Ø36	$24	$	$	$	$
Ø37	$25	%	%	%	%
Ø38	$26	&	&	&	&
Ø39	$27	'	'	'	'
Ø4Ø	$28	(	(	(	(
Ø41	$29	)	)	)	)
Ø42	$2a	*	*	*	*
Ø43	$2b	+	+	+	+
Ø44	$2c	,	,	,	,
Ø45	$2d	—	—	—	—
Ø46	$2e	.	.	.	.
Ø47	$2f	/	/	/	/
Ø48	$3Ø	Ø	Ø	Ø	Ø
Ø49	$31	1	1	1	1
Ø5Ø	$32	2	2	2	2
Ø51	$33	3	3	3	3
Ø52	$34	4	4	4	4
Ø53	$35	5	5	5	5
Ø54	$36	6	6	6	6
Ø55	$37	7	7	7	7
Ø56	$38	8	8	8	8
Ø57	$39	9	9	9	9
Ø58	$3a	:	:	:	:
Ø59	$3b	;	;	;	;

060	$3c	<	<	<	<
061	$3d	=	=	=	=
062	$3e	>	>	>	>
063	$3f	?	?	?	?
064	$40			@	@
065	$41		A	A	a
066	$42		B	B	b
067	$43		C	C	c
068	$44		D	D	d
069	$45		E	E	e
070	$46		F	F	f
071	$47		G	G	g
072	$48		H	H	h
073	$49		I	I	i
074	$4a		J	J	j
075	$4b		K	K	k
076	$4c		L	L	l
077	$4d		M	M	m
078	$4e		N	N	n
079	$4f		O	O	o
080	$50		P	P	p
081	$51		Q	Q	q
082	$52		R	R	r
083	$53		S	S	s
084	$54		T	T	t
085	$55		U	U	u
086	$56		V	V	v
087	$57		W	W	w
088	$58		X	X	x
089	$59		Y	Y	y
090	$5a		Z	Z	z
091	$5b			[	[
092	$5c			£	£
093	$5d			]	]
094	$5e			↑	↑
095	$5f			←	←
096	$60				
097	$61				A
098	$62				B
099	$63				C
100	$64				D
101	$65				E
102	$66				F
103	$67				G
104	$68				H
105	$69				I
106	$6a				J
107	$6b				K
108	$6c				L
109	$6d				M
110	$6e				N
111	$6f				O
112	$70				P
113	$71				Q
114	$72				R
115	$73				S
116	$74				T
117	$75				U
118	$76				V
119	$77				W
120	$78				X
121	$79				Y
122	$7a				Z
123	$7b				
124	$7c				
125	$7d				
126	$7e				
127	$7f				
128	$80				Keine Funktion
129	$81				Keine Funktion
130	$82				Keine Funktion
131	$83				Keine Funktion

132	$84			Keine Funktion	
133	$85			Weiss	
134	$86			Keine Funktion	
135	$87			Keine Funktion	
136	$88			Shift Commodore verboten	
137	$89			Shift Commodore erlauben	
138	$8a			Keine Funktion	
139	$8b			Keine Funktion	
14Ø	$8c			Keine Funktion	
141	$8d			Return	
142	$8e			Kleinbuchstaben	
143	$8f			Keine Funktion	
144	$9Ø			Keine Funktion	
145	$91			Cursor abwaerts	
146	$92			Revers anschalten	
147	$93			Cursor Home	
148	$94			Delete	
149	$95			Keine Funktion	
15Ø	$96			Keine Funktion	
151	$97			Keine Funktion	
152	$98			Keine Funktion	
153	$99			Keine Funktion	
154	$9a			Keine Funktion	
155	$9b			Escape	
156	$9c			Rot	
157	$9d			Cursor rechts	
158	$9e			Gruen	
159	$9f			Dunkelblau	
16Ø	$aØ			Space	
161	$a1			!	!
162	$a2			"	"
163	$a3			#	#
164	$a4			$	$
165	$a5			%	%
166	$a6			&	&
167	$a7			'	'
168	$a8			(	(
169	$a9			)	)
17Ø	$aa			*	*
171	$ab			+	+
172	$ac			,	,
173	$ad			—	—
174	$ae			.	.
175	$af			/	/
176	$bØ			Ø	Ø
177	$b1			1	1
178	$b2			2	2
179	$b3			3	3
18Ø	$b4			4	4
181	$b5			5	5
182	$b6			6	6
183	$b7			7	7
184	$b8			8	8
185	$b9			9	9
186	$ba			:	:
187	$bb			;	;
188	$bc			<	<
189	$bd			=	=
19Ø	$be			>	>
191	$bf			?	?
192	$cØ			@	@
193	$c1			A	a
194	$c2			B	b
195	$c3			C	c
196	$c4			D	d
197	$c5			E	e
198	$c6			F	f
199	$c7			G	g
2ØØ	$c8			H	h
2Ø1	$c9			I	i
2Ø2	$ca			J	j
2Ø3	$cb			K	k

Dez	Hex				
2Ø4	$cc			L	l
2Ø5	$cd			M	m
2Ø6	$ce			N	n
2Ø7	$cf			O	o
2Ø8	$dØ			P	p
2Ø9	$d1			Q	q
21Ø	$d2			R	r
211	$d3			S	s
212	$d4			T	t
213	$d5			U	u
214	$d6			V	v
215	$d7			W	w
216	$d8			X	x
217	$d9			Y	y
218	$da			Z	z
219	$db			[	[
22Ø	$dc			£	£
221	$dd			]	]
222	$de			↑	↑
223	$df			←	←
224	$eØ				
225	$e1				A
226	$e2				B
227	$e3				C
228	$e4				D
229	$e5				E
23Ø	$e6				F
231	$e7				G
232	$e8				H
233	$e9				I
234	$ea				J
235	$eb				K
236	$ec				L
237	$ed				M
238	$ee				N
239	$ef				O
24Ø	$fØ				P
241	$f1				Q
242	$f2				R
243	$f3				S
244	$f4				T
245	$f5				U
246	$f6				V
247	$f7				W
248	$f8				X
249	$f9				Y
25Ø	$fa				Z
251	$fb				
252	$fc				
253	$fd				
254	$fe				
255	$ff				

Anhang -D- Quellennachweise

Procedural Elements for Computer Graphics
David F. Rogers
ISBN 0-07-053534-5
McGraw-Hill Book Company

Advanced Programming Techniques for the BBc Micro
Jim McGregor & Alan Watt
ISBN 0-201-14059-4
Addison-Wesley Publishing Company

CAD-Technik
Günter Spur & Frank-Lothar Krause
ISBN 3-446-13897-8
Carl Hanser Verlag

Numerische Probleme und ihre Lösung mit Taschenrechnern
Jon M. Smith
ISBN 3-528-08380-8
Friedr. Vieweg & Sohn

Computer Graphics for the IBM Personal Computer
Donald Hearn & M. Pauline Baker
ISBN 0-13-164335-5
ISBN 0-13-164327-4
Prentice-Hall Inc.

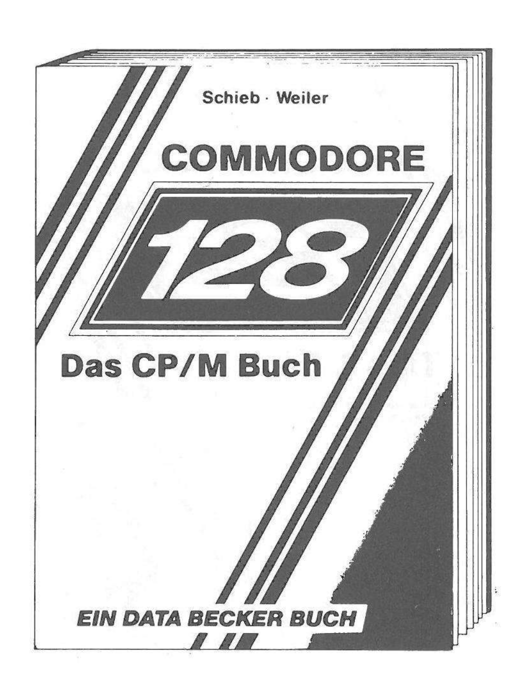
Das Grafikbuch zum Commodore 64
Axel Plenge
ISBN 3-89011-011-8
Data Becker GmbH

Commodore 128 Intern
Gerits, Schieb, Thrun
ISBN 3-89011-098-3
Data Becker GmbH

Einführung in CAD mit dem Commodore 64
Heift
ISBN 3-89011-067-3
Data Becker GmbH

Das Maschinensprachebuch für Fortgeschrittene zum Commodore 64
Lothar Englisch
ISBN 3-89011-022-3
Data Becker GmbH

64 Intern
Angerhausen, Brückmann, Englisch, Gerits
ISBN 3-89011-000-2
Data Becker GmbH



Falls Sie auf dem Commodore 128 das CP/M einsetzen wollen, sollten Sie dieses Buch lesen! Von grundsätzlichen Erklärungen zur Speicherung von Zahlen, Schreibschutz oder ASCII, Schnittstellen und Anwendung von CP/M-Hilfsprogrammen. Für Fortgeschrittene: CP/M und Commodore-Format, Erstellen von Submit-Dateien u.v.m. Nutzen Sie die vollen Möglichkeiten des Standard-Betriebssystems CP/M!

Weiler/Schieb
Das CP/M-Buch zum C-128
340 Seiten, DM 49,–
ISBN 3-89011-116-5



Jetzt gibt es das große Floppybuch auch zur 1570/1571! Mit einer Einführung für Einsteiger, Arbeiten mit dem C-128 und BASIC 7.0, einer umfassenden Einführung in das Arbeiten mit sequentiellen und relativen Dateien, Programmierung für Fortgeschrittene: Nutzung der Direktzugriffsbefehle, Programme im DOS, wichtige DOS-Routinen und ihre Anwendung und natürlich ein ausführlich dokumentiertes DOS-Listing.

Ellinger
Das große Floppybuch zur 1570/1571
583 Seiten, DM 49,–
ISBN 3-89011-124-6

LADEN, STARTEN - KLAR !

Für alle diejenigen, die sich fleißiges Abtippen ersparen und trotzdem alle Programme nutzen wollen, gibt es einen Ausweg:

Mit der nebenstehenden Post-Zahlkarte einfach die Diskette zum Buch bestellen !

Diskette zum Buch „Das große Grafik-Buch zum C 128"
DM 29,--



Empfängerabschnitt

DM 29,-- Pf ---

für Postgirokonto Nr. 789 - 436

Absender (mit Postleitzahl)

Verwendungszweck
Diskette z. Buch
376 154

Zahlkarte (Mit Schreibmaschine, Tinte oder Kugelschreiber deutlich ausfüllen)

DM 29,-- Pf ---

(DM-Betrag in Buchstaben wiederholen) Neunundzwanzig

für DATA BECKER GmbH
Merowingerstr. 30
in 4000 Düsseldorf

Postgirokonto Nr. 789 - 436

Postgiroamt Essen

Postvermerk

Absender:

DM 29,-- Pf ---

für Postgirokonto Nr. 789 - 436

Einlieferungsschein
– Bitte sorgfältig aufbewahren –

DM 29,-- Pf ---

für DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Postgirokonto Nr. 789 - 436

Postvermerk

Für Vermerke des Absenders